AN IMPLEMENTATION OF A CODASYL BASED DATA BASE MANAGEMENT SYSTEM UNDER THE UNIX OPERATING SYSTEM

John Edward Howard

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN IMPLEMENTATION OF A CODASYL BASED
DATA BASE MANAGEMENT SYSTEM
UNDER THE UNIX OPERATING SYSTEM

by

John Edward Howard

June 1976

Thesis Advisor: G. L. Barksdale, Jr.

Approved for public release; distribution unlimite

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) An Implementation of a CODASYL Based Data Base Management System under the UNIX Operating System		5. TYPE OF REPORT & PERIOD COVERED Master's thesis; June 1976
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) John Edward Howard		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1976
		13. NUMBER OF PAGES 167
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Data base
Data Base Management System
Network model
CODASYL DBTG
UNIX

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis reports the implementation of a Data Base Management System (DBMS) based on the CODASYL design. The DBMS was implemented on a DEC PDP 11/50 computer utilizing the UNIX operating system. Background material includes a discussion of data base history and techniques, design of UNIX and the C programming language. The research performed was the adaptation of the CODASYL DBMS design to the UNIX environment and the design

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
(Page 1) S/N 0102-014-6601 |

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (cont.)

of a C language Data Description Language (DDL) and Data Manipulation Language (DML) to interface the DBMS to user programs. Conclusions and recommendations for improvements are also included.

An Implementation of a CODASYL Based
Data Base Management System
under the
UNIX Operating System

by

John Edward Howard
Captain, United States Marine Corps
B. A., University of Texas at Austin, 1969

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
JUNE 1976

ABSTRACT

This thesis reports the implementation of a Data Base Management System (DBMS) based on the CODASYL design. The DBMS was implemented on a DEC PDP 11/50 computer utilizing the UNIX operating system. Background material includes a discussion of data base history and techniques, design of UNIX and the C programming language. The research performed was the adaptation of the CODASYL DBMS design to the UNIX environment and the design of a C language Data Description Language (DDL) and Data Manipulation Language (DML) to interface the DBMS to user programs. Conclusions and recommendations for improvements are also included.

CONTENTS

LIST OF FIGURES

I. INTRODUCTION.

The Conference on Data Systems Language (CODASYL) has defined a data base system [Ref. 2 and 3] which is partially generalized and partially tailored to COBOL. This system is based on network data modeling techniques. CODASYL has made the claim that the system could have other languages effectively interfaced to it and that the system could be implemented in a variety of environments. The Computer Science Department has two Digital Equipment Corporation PDP 11/50 computers running with the UNIX operating system [Ref. 1]. The equipment was acquired for research in signal processing applications. This environment is one in which a CODASYL based data base management system has never been introduced.

Relational data modeling techniques are the major competitor with the CODASYL system. Recently, development was completed on a relational data base management system which runs under UNIX [Ref. 4 and 5]. This system is called INGRESS and was developed at the University of California at Berkeley. Currently a discussion is taking place in the literature over the relative merits and drawbacks of relational models versus network models (chiefly the CODASYL version) [Refs. 6, 7, 8 and 9]. Although much has been written about the merits of each model, relatively little empirical comparison has been done. Therefore, since steps

are being taken to aquire the INGRESS software, it was decided that a CODASYL data management system would provide a complete suite of data management software.

The tasks to be accomplished were design and implementation of a UNIX hosted CODASYL system, design and implementation of a C language [Ref. 10] interface to this system, aquisition of INGRESS and comparative studies of the two systems for signal processing applications. This thesis documents the design and implementation of a UNIX hosted CODASYL data base management system and the design of a C language interface to this system.

II. BACKGROUND.

A. Data Access Methods - A History.

1. Technological Effects.

During the first and second generations of computer hardware, data storage media were tapes, relatively slow disks and drums and the omnipresent punched card. Data storage and retrieval concepts were shaped by these devices, especially the punched card. A file was therefore a sequentially ordered and accessed, contiguously stored group of records. All the records were of fixed length. This view is also oriented toward a monoprogramming environment with absolute separation of one user's files from another's.

With the advent of third generation technology, several factors began to affect data storage and retrieval concepts. Foremost was the development of fast, high capacity, relatively inexpensive direct access storage devices. These devices stimulated the development of a whole range of new access techniques such as hash coded and indexed file organization. Secondly, the multi-user environment caused a breakdown of the sharp division between the execution environment of users. This breakdown was accompanied by a rethinking of the relationships between the overlapping data requirements of users. All these factors led to the

development of a new kind of file system known as a data base.

2. Access Methods.

The inspiration for a multi-purpose data base came from management systems in which it was discovered that vast overlap and duplication of data was occuring between different groups in a company. For example the payroll and personnel sections would typically each have employee files which were stored and maintained separately but which overlapped by 80 per cent in data content. An early management information system (MIS) which attacked this problem was IBM's Bill of Material Program (BOMP) [Ref. 12] which allowed the structuring of a parts list with subassemblies each having its own parts lists; this facilitated the management of manufacturing inventories. When the subassemblies occured in many different parts, the savings afforded through avoided data duplication were significant. The BOMP used a relatively flexible list structure and marked a significant departure from traditional file organization.

With the advent of the consolidated multi-purpose data base, a whole new level of data structuring was imposed on the techniques for physically mapping files to devices. These data structures employed relatively complex methods from graph theory and other disiplines which had previously been used only on relatively small amounts of data residing in main storage.

This new level of data base structure combined with the fact that a very large portion of the data base is typically on-line, has imposed on programmers the requirement for a new skill. This skill has been called navigation through the data base [Ref. 13]. In this view, a programmer must travel via access paths through the data base searching for landmarks until he has located the data he desires. Choosing an inappropriate access path can be extraordinarily inefficient and costly in time, so the penalty for lack of navigational skill is high. It would obviously be desirable to remove as much of the burden for navigation from the programmer as is practical. The developement of modern data base management systems (DBMS) has been made difficult by the dilemma of desiring both optimal access paths and ease and simplicity of use for the programmer.

3. Terminology.

This section will attempt to define the terms used in this field of inquiry. The following definition of a data base is due to Ref. 13:

> "A data base may be defined as a collection of interrelated data stored together with as little redundancy as possible to serve one or more applications in an optimal fashion; the data are stored so that they are independent of programs which use the data; a common and controlled approach is used in adding new data and in modifying and retrieving existing data within the data base. One system is

said to contain a collection of data bases if they are entirely separate in structure."

Two types of languages are mentioned in connection with DBMS. The first is the Data Description Language (DDL) which describes the types of data entities which may exist along with the allowable attributes. There may be two DDL's or two levels of DDL for describing a data base. The first level description is the system's view of the data base as it is actually organized and the second, a user's view of the data base. These levels are called the schema and sub-subschema respectively. In the relational model terminology, the DDL may be called the relational algebra.

The second language is the Data Manipulation Language (DML) which is concerned with the storage, retrieval and modification of specific occurences of the entity types described by DDL statements. In relational model terminology, this language corresponds to the relational calculus. The entities handled by DDL and DML may be records, sets or anything that may need manipulation. The attributes may be such things as data items, set membership, set ownership or location within the data base.

The data base model is the meta-structure which is imposed on the organization of the data base. The model prescribes the types of entities which are allowed. It defines the data attributes and structural attributes that an entity may have. The definition of a DDL and DML is the

implementation of the meta-structures of a data base model. Currently the two most widely discussed models are the network model and the relational model.

4. Goals of a DBMS.

The following goals have been proposed for a DBMS [Ref. 3].

- Allow the data structures suited to each particular application while permitting multiple applications to use the data without need for data redundancy.

- Allow more than one process to concurrently retrieve or update data in the data base.

- Enable the use of a variety of search strategies against an entire data base or a portion of it.

- Provide protection of data from unauthorized access.

- Provide centralized control over the placement of data.

- Provide device independence for programs.

- Allow the user to interact with the data but be free of the mechanics of maintaining the structural associations which have been declared.

- Allow as great an independence of programs from data and structures as possible.

- Make the data description independent of any particular programming language but give it the capability of interfacing with a variety of programming languages.

These goals seem to be generally agreed on in the literature as being reasonably complete. There is considerable controversy, however, over the relative importance of individual goals. In particular, some contend that the primary goal should be to allow the user to be free of the data base structures entirely [Ref. 9].

B. The Network Model.

One of the two data base models which has received wide attention is the network model. This model is grounded in graph theory and relationships between data are represented by some form of directed graph. The nodes of the graph may be entities containing data attributes or may simply be place holders whose only attributes are the arcs of the graph. The arcs represent logical links between the entities which can be travelled in the direction of the arc to navigate through the data base. Thus, even though the implementation of the arcs may be transparent to the user, the access paths are visible to the user as part of the structure of the data base. The DML is said to be prescriptive of the data access paths, that is, it must prescribe the course through the data structures.

Various restrictions as to the type of network allowed may be imposed on a network model. For example, the graphs may be required to be acyclic or the structures may be restricted to trees, chains or lists. A non-homogeneous model has the restriction that if two nodes are connected by an

arc then the entities represented by the nodes may not be of the same type. A special class of the non-homogeneous network model is the hierarchical model. Hierarchical models have the following restrictions: the graphs must be trees, an entity of any type may appear only once on a particular branch of the tree and certain entity types must always appear on a given branch at a higher level than other entity types. An example of a hierarchical data base would be one with the entities country, state, county and city. For a particular country, one or more of the entities of state and county may be left out between a country and its cities, but a city cannot appear above a state, nor can it appear above another city. An example of a non-hierarchical data base is the BOMP in which a subassembly may contain other subassemblies which may in turn contain subassemblies. Note that the BOMP is homogeneous since subassemblies are linked to subassemblies.

C. The Relational Model.

The second of the two most discussed modes for data representation is the relational model. This model is grounded in set theory and specifically in the concept of a relation in the mathematical sense. Given sets S1, S2, ..., Sn (not necessarily distinct), R is a relation on these sets if it is a sub-set of the Cartesian product of S1 x S2 x ... x Sn. The element of R are n-tuples whose jth component is from Sj, for j from one to n. R is said to be an n-ary

relation or of degree n and Sj is called the jth domain of R.

In the relational model, each set Sj must be a set of like-type attributes. The relations are named and time variant according to some maintenance algorithm. An example of a relation would be a set called time-spectrum made up of frequency, time and amplitude triples. This ternary relation might represent the latest 30 minutes of data from a hydrophone. Every relation must have a key by which the tuples can be identified. The key must be unique (no two tuples with the same key) and non-redundant (the whole key is needed for identification). In the above example, frequency and time make up the key.

The chief advantage of the relational model is that the user's view of the data is independent not only of the physical mapping to media, but also of the access paths involved. The chief disadvantage is the difficulty of devising an implementation which is reasonably efficient for all applications [Ref. 8].

A good deal of work has been done on normalizing relations to remove undesirable data representational characteristics and providing appropriate operations and transformations for relations. References 14, 15 and 16 give a definitive exposition on the theory of relational data models.

D. The CODASYL DBTG DBMS.

1. History.

CODASYL is an informal and voluntary organization of interested individuals, supported by their institutions, who contribute their efforts and expenses towards the ends of designing and developing techniques and languages to assist in data systems analysis, design and implementation. Founded in 1959, its most famous achievement has been the definition of the COmmon Business Oriented Language (COBOL). In June, 1965, the CODASYL COBOL Language Subcommittee of the Programming Languages Committee (PLC) resolved to organize a task force to study list processing. In November, 1965, this task force produced a proposed list processing extension to COBOL for file management. In May, 1967, the List Processing Task Force changed its name to the Data Base Task Group (DBTG) and undertook a comparative study of data base management techniques and systems. This study was culminated by the publication of an interim report in February, 1968 and the agreement by the Language Subcommittee that "COBOL needs the Data Base Concept" [Ref. 2]. At the Tenth Anniversary Meeting of CODASYL held in May, 1969, consideration was given to separating the data description and data manipulation languages. The idea received wide endorsement at the meeting and was the basis for the direction of efforts by the DBTG until October, 1969 at which time Ref. 17 was presented. From the time of publication of Ref. 17 until the publication of Ref. 3 in April, 1971, 179

proposals for changes and extensions to the DDL and DML were considered, of which 130 were incorporated into Ref. 3. In June, 1971, it was decided that the schema DDL should be developed separately from the COBOL DDL and DML. Accordingly, the Data Description Language Committee (DDLC) was formed as a separate organization from the PLC. The DDLC proceded with modifications and enhancements to the DBMS and schema DDL definitions and, in June, 1973, produced Ref. 2. This document is currently the basis for the CODASYL DBMS.

2. Terminology and Concepts.

For a complete description of the CODASYL schema DDL statements and DBMS design see Ref. 2. The schema DDL is used to describe a data base and has the following entity types: Data items, data aggregates, records, areas and sets.

A data item is an occurence of a named atomic data attribute. It is the smallest unit of named data. The set of values that a data item can assume is called its range. The range of an item is always restricted to values of a particular type. The possible types are arithmetic data, string data, data base keys and implementor defined types.

A data aggregate is an occurence of a named collection of data items. There are two kinds: vectors and repeating groups. A vector is a one dimensional sequence of data items, all with identical characteristics. A repeating group is a collection of data attributes that occurs

multiple times within a record occurence. The collection of attributes may include data items and data aggregates.

A record is an occurence of a named collection of zero or more data items or data aggregates. Each record entry defines a record type of which there may be zero or more occurences within the data base. The record is the smallest addressable entity within the data base.

A set is a named collection of records. Each set entry in the schema defines a set type for which zero or more occurences (sets) may exist in the data base. Each set type declared in the schema must have one record type declared as its owner and may have one or more record types declared as its members. Each set occurence which exists in the data base must contain exactly one record of its owner type and zero or more of its member record types. A special set type may be delared which has one and only one occurence and whose owner is the DBMS. A set so declared is said to be a singular set. There is no provision for a record type to be both an owner and member record type of the same set. This means the CODASYL model is non-homogeneous. It is not, however, hierarchical in that set types may be defined with ownership and membership such that cycles can occur.

An area is a named collection of records which need not preserve owner/member relations. An area may contain occurences of multiple record types and a record type may occur in multiple areas. A particular record occurence of a

record is assigned to an area when it when it is created and it may not migrate out of that area. An area may be declared to be temporary. Temporary areas are created especially for a run-unit, exist for the life of the run-unit and are destroyed when the process terminates. Two run-units may have a particular temporary area open concurrently but each run-unit is using a different version of the area which is unique to that particular run-unit. The concept of area allows the subdivision of the data base. It allows the DBMS to control placement of an entire area to provide efficient storage and retrieval. Areas are a convenient unit for recovery and also provide a convient, natural subdivision for allowing a part of the data base to be removed to off-line storage.

A schema consists of DDL entries and is a complete description of a data base. It includes the names and descriptions of all areas, set types and record types that may appear in the data base. A data base is the totality of all records, sets and areas controlled by a schema. For an installation to have multiple data bases, it must have multiple schemas and the content of the data bases must be disjoint.

No schema DDL entry may include references to the physical devices or media space. Thus a schema written in the DDL is independent of the physical storage of data and the data may be stored on any combination of storage media available to a DBMS. Some devices, due to their sequential

nature, may not allow the full advantages of DDL facilities, however the use of these devices is not precluded.

A program is a set or group of instructions. User programs must have access to a sub-schema DDL description of that portion of the data base they are interested in. Additionally, they must be able to use a DML to interact with the data base through the DBMS.

A run-unit is the execution of one or more programs viewed by the operating system as a unit. Under OS/360, the run-unit might be a job and under UNIX, a parent process and any children. The run-unit makes requests of the DBMS which in turn consults the schema and interacts with the operating system to fulfill the request.

A user working area (UWA) is conceptually a loading and unloading zone where all data provided to a run-unit by the DBMS and all data to be picked up by the DBMS must by placed. The DBMS has its own system buffers which it uses to manipulate the data base. It uses the UWA only for input and output of data for the requesting run-unit. Each run-unit has its own UWA.

3. The Schema vs. the Sub-schema.

The subschema has the following characteristics. An arbitrary number of possibly overlapping sub-schemas may be declared. Multiple programs may reference a sub-schema but they have access only to that portion of the data base

included in the sub-schema. Thus, the sub-schema DDL description enables the subsetting of the data base so that a user program need only worry about that portion of the data base it uses, and insulates the remainder of the data base from the user. A measure of the data independence is provided between the schema and sub-schema. The sub-schema description may differ from the schema in the following ways.

a. Data Item Level.

Descriptions of items may be omitted. Included items may be of a different type or in a different position within the record.

b. Data Aggregate Level.

Descriptions of specific data aggregates may be omitted. Data aggregates and items may have additional structure imposed on them (e.g. vectors may become multi-dimensional arrays). The position of data aggregates within a record may be changed.

c. Record Level.

Descriptions of records may be omitted. Descriptions of new record types composed of data from other record types may be introduced (not supported by the COBOL or C DDL's).

d. Set Level.

Descriptions of specific set types may be omitted. Different set selection criteria may be specified. Descriptions of specific member-record types may be omitted.

e. Area Level.

Descriptions of specific areas and the records within them may be omitted, while occurences of the same record type in other areas are included.

4. The Schema and the DML.

The relationship between the DDL and the DML is that between declarations and procedures. In order to specify this relationship, a set of basic data manipulation functions must be defined which is DML and host language independent. Specific commands provided by a particular DML must be resolved into these basic functions. Basic functions include the capability of selecting records, presenting them to a run-unit and adding, changing or removing records and relationships.

5. Data Base Administration.

Certain facilities must be availible to support the user programs. These tools are not defined in the CODASYL DBMS and may include the following.

a. Recovery Routines.

Data base recovery routines may be used including activity logging, checkpoint and rollback.

b. Utility Routines.

Utility and service routines are required to support a data base in day-to-day operations. Examples include routines for editing and printing, loading and dumping, preconditioning, garbage collection, statistical analysis and comparison.

c. Schema Meta-language.

This language permits changes in the schema and cause them to be reflected in the data base. Without such a language, the changes must be made by defining a new schema and recreating the data base accordingly.

d. Device Media Control Language (DMCL).

This language provides for assignment of data to devices and media space, and specification and control of buffering, paging and overflow.

6. Data Base Procedures.

At various points in the accessing of a data base, non-standard computations or processing may be required. To allow for these situations, the capability is provided to define data base procedures. These procedures may be

invoked for checking of privacy locks, producing computed results from other items, searching algorithms, data compression and expansion, validity checking or system instrumentation.

7. Record Placement Control.

The schema DDL permits specification of an area or areas to which record occurences of a particular type must be assigned. The schema DDL also includes a clause which causes records being added to be placed near some other record. Conceptually, the effect of such clauses is to cause clustering of records which are likely to be used in conjunction with one another. These declarations for selecting the area and location within the area are the WITHIN clause and the LOCATION clause, respectively, of the record subentry. The fact that the schema DDL permits placement control is not assumed by CODASYL to have any physical connotations.

8. Data Base Keys.

The DDL assumes that every record occurence in the data base has a unique identifier which enables the DBMS to distinguish it from every other record in the data base. This key must be assigned when the record is created and remains with it for the life of the record. This key may be supplied to the DBMS by a run-unit or data base procedure, generated from the record's contents or assigned by the DBMS. The permanence of the key must be insured since any

run-unit may use the keys to refer to the record.

9. Ordering of Sets.

Each set type declared in the schema must have an ordering specified for it. This order is maintained by the DBMS and is a logical, not a physical, ordering. Thus, the same record occurences could participate as members in several sets of different types and be ordered differently in each of the sets. The member records of each occurence of a given set type can be ordered in any of the following ways.

- Sorted in ascending or descending order based on the value of specified keys. These keys may be data items in the member records, the names of the member records, the data base keys of the member records or some combination of the above.

- Sorted in the order resulting from inserting new member records first in the set, last in the set or before or after the set member which is currently known to the requesting run-unit.

- Sorted in the order most convenient to the DBMS.

10. Search Keys.

An arbitrary number of search keys may be declared for a set type regardless of whether it is sorted or not. The components of the search keys must be data items included in the member records of the set. The declaration of a search key causes the DBMS to develop and use some kind

of indexing for the member records of each occurence of the set type. The term indexing is used here to refer to any technique which does not involve a complete scan of the records involved.

11. Set Membership.

A record type may have different kinds of set membership declared for different set types. Automatic membership means that membership is established in an appropriate occurence of a set type when a record is added to the data base. Manual membership means that membership can only be established in a set occurence by a run-unit executing an insert function.

Mandatory or optional membership concerns the removal of a record from a set occurence. Once a record has been established as a member of a set for which it has mandatory membership, it cannot be removed until the record is deleted. If the membership is optional, the record may cease to have membership via a remove function.

A set type may be declared as dynamic. A dynamic set may have a record of any type inserted into it or removed from it. If a set type is declared to be dynamic, no member records may be declared for it.

12. Set Selection.

In general, there will be more than one set of a given type in the data base. It is therefore necessary to

provide a means for identifying the proper set when member records are stored and retrieved. The SELECTION clause of the member subentry in the DDL controls the strategy for selecting a specific set of a given type. A separate SELECTION clause is required for each member record type and set type pair. The SELECTION clause provides for naming a series of sets which form a continous path to the desired set. For all the sets along the path, other than the first named set, the DBMS limits its search to the member records of the set selected at the previous step in the path.

13. Privacy of Data.

Protection against unauthorized data access is provided through a mechanism of privacy locks which are specified in the schema. Privacy keys must be provided by a run-unit seeking to access or alter data protected by a privacy lock. The schema DDL provides for declaring privacy locks at the schema, area, record, data item, data aggregate, set and member levels. Locks can be declared for specific functions at each of these levels. A privacy lock is either a value which must be matched by a corresponding privacy key or a data base procedure which is called to validate the privacy key. If a procedure is used, it returns a yes or no answer, and beyond this the action of such a procedure is implementor defined.

14. Integrity of Data.

The DDL provides for the checking of the validity of a data item whenever a value is changed or a new value is stored in the data base. In addition, provision is made for the naming of data base procedures which the DBMS invokes when a run-unit attempts to update nominated records or sets. This feature enables a check of any update or series of updates applied to the data base.

E. Laboratory Equipment and Software.

The computer equipment in the laboratory consists of two PDP 11/50's with associated peripherals. The information about the equipment which is relevant to this thesis is minimal, however it should be noted that the DBMS was developed using an interactive display terminal and is oriented toward that environment.

The operating system which supports the DBMS is UNIX. Reference 1 contains a supplement to the following discussion of UNIX.

1. The File System.

The most important function of UNIX is to provide a file system. From the user's point of view their are three kinds of files: ordinary files, directories and special files.

An ordinary file can contain any information the user desires. The system imposes few special structure requirements on files, however some programs expect files of a certain format. A text file consists of a string of characters with lines delimited by new line characters. A binary program file is a sequence of words as they will appear in main memory when the program is executed. The assembler and loader programs use special object file formats.

Directories provide the mapping between the names of files and the files themselves. They induce a structure on the file system as a whole. A directory behaves exactly like an ordinary file except that the system controls its format and contents. Each system user has a directory associated with his user name and he may create sub-directories to organize collections of his files. The system has several directories which it maintains for its own use. One is the root directory. The directories in a file system form a tree and the root is the base of this tree. Thus, any file in the system can be located by tracing a path from the root through the appropriate directories. Another system directory contains all the programs which are used as system commands and is special only in that certain programs "know" its name.

Each directory must appear as an entry in exactly one other directory called its parent. Each directory has two special entries. These are the name "." which refers to

the directory itself and the name ".." which refers to the parent directory. These entries enable reference to the directory and its parent without knowing the name explicitly.

File names are strings of 14 or fewer characters. Identification of a file to the system is accomplished through a string of directory names separated by virgules ("/") and terminated by the file name desired. This string is called a path name. When the path name is started with a virgule, the system begins the path search at the root directory, otherwise it starts at the user's current working directory. For example, the path name "/foxtrot/uniform/charlie" would cause the system to start at the root, search for directory "foxtrot", search "foxtrot" for directory "uniform" and find file "charlie" in "uniform". The file "charlie" could be any type file, including a directory. In another case, the pathname "kilo" would cause the system to search the user's current directory for "kilo". The path name "/" refers to the root itself.

Special files provide the means of handling I/O devices. Each device supported by UNIX, including communications lines and main memory, is associated with one or more special files. These files can be read or written in the same manner as ordinary files except that the result is the activation of the appropriate device. All special files reside in directory "/dev".

The access control or protection scheme in UNIX is relatively simple. Each user known to the system has a unique user number called the user id. When a file is created, the appropriate user id is associated with it and bits are set in the directory entry indicating which users have permission to read, write or execute the file. A facility is provided for executable files called set-user-id whereby when the files are executed the resulting process assumes the user id of the owner of the executable file. This enable a system program executed by a user to access files which the user cannot directly access himself. Since anyone may cause his executable files to use set-user-id, this feature is generally available to provide protected access to files. The system recognizes one user id (the "super user") as being free of any access restrictions. The major flaw in the UNIX protection scheme is that there is no way to monitor or lock out simultaneous opening of a file by multiple programs with access rights to the files. The system's authors contend that these features are neither necessary nor sufficient for integrity controls [Ref. 1]. However, the reasoning behind declaring the features unnecessary was that "we are not faced with large single-file data bases maintained by independent processes".

2. Input/Output (I/O) Calls.

Under UNIX, I/O calls are designed to eliminate the difference between the various devices and forms of access. The file system organizes all media space into 512 byte

blocks which are its smallest readable and writable unit. Consequently, reads and writes of 512 bytes starting on a 512 byte boundary are most efficient. However, no logical record size is imposed by the system, nor is there any distinction between random or sequential access. To read or write an already existing file, an "open" call must be made. This system call is passed a path name and returns a number, called a file descriptor, which identifes the open file to the system. The file descriptor is used in subsequent I/O calls. In order to create and open a file, a "creat" call must be made. This call requires parameters which specify the file name and access mode, and returns a file descriptor. A "creat" on an existing file truncates it to zero length. An open file may be accessed via "read" and "write" calls. These system calls require the file descriptor, the location of a read/write buffer and the length of the buffer.

To enable random access of appropriate files, the "seek" call is provided. This system call merely changes the read/write pointer associated with an open file. The read/write pointer contains the byte offset from the beginning of the file at which the next access will begin. Other system calls exist for such file manipulations as closing a file, finding the status of a file, changing the protection mode or owner of a file, creating or removing a directory, making a link to an existing file and deleting a file.

3. Processes and Images.

An image is an entire computer execution environment including main memory image, general register values, the status of open files and the identity of the current directory. Thus the image constitutes a state vector of a process which contains all information necessary to resume execution of the process. A process is the execution of an image while the virtual machine is imposed on the hardware by the system. The virtual address space of a process is divided into three logical segments: the program text (instructions and constants), data and stack. Pure text is read only for the user while the data and stack segments may expand or contract in size.

A process comes into existence through a "fork" call executed by another process. This system call creates an exact duplicate of the image of the calling process. The only difference between the processes is that one process is considered the parent and the other the child. Both execute as if returning from the "fork" call. The parent receives as a return value a number called the process id, which uniquely identifies the child.
The child receives zero as its return value. Synchronization between parent and child is provided by the "wait" call. When a process with children executes a "wait", its execution is suspended until one of its children terminates. The return value of the "wait" is the process id of the terminated child. Interprocess communication is provided by

the "pipe" call. This system call sets up a channel which can be read or written by any process which has as an ancestor the process that executed the "pipe" call.

The "exec" system call is provided to allow the execution of a program (i.e., executable file). The "exec" call needs a path name to the file as its argument. A process executing an "exec" has all its code, data and stack space overlaid by the referenced program if the call succeeds. Open files, the current directory and interprocess relationships remain unchanged. A return from the "exec" occurs only if the function is unsuccessful. Termination of a process can be accomplished via an "exit" system call. When an "exit" is executed, the process and associated image cease to exist.

4. The C Language.

C is the programming language primarily used under UNIX. Most of UNIX itself is coded in C. C provides modern control structures to allow structured GOTO-less coding. Its design objectives were to give shorter and clearer code, encourage modularity and good program organization and provide facilities for many different types of data including pointers and character strings.

A C program consists of a group of functions (one of which must be named "main") and possibly some external data declarations. Parameters may be passed between functions via call and return arguments or through external data

items. C is not a block structured language in that functions cannot be defined locally to other functions and external data names may not be redeclared locally to a function. However, the block structured language feature of allowing a group of statements to be considered as a single statement is included. This grouping is accomplished by enclosing the statements within "{" and "}".

The basic data types in C are "int", "char", "float", "double" and "struct". In addition, arrays of or pointers to any of these types can be declared. Items of type "int" are 16-bit two's complement integers. Items of type "char" are 8-bit values which can be interpreted as characters or as two's complement integers. Strings are represented as arrays of characters. Items of type "float" or "double" are binary floating point numbers of length 32 and 64 bits repectively. An item of type "struct" consists of a group of item declarations (possibly including arrays) which can be viewed as a unit. This latter capability provides for user definition of a theoretically infinite number of data types.

C provides a large number of binary and unary arithmetic and logical operators. Arithmetic operations provided are addition and subtraction, multiplication and division, incrementation and decrementation, and bit-wise OR, AND and complement. Logical operators allow expressions to be compared, logically AND'ed, logically OR'ed and logically complemented. No distinction exists between a logical

expression and an arithmetic expression. Any expression has a true value if and only if it evaluates to a non-zero value.

Assignment statements are provided in C which are unusual in the following ways. An assignment statement can be used as an expression and has the value that was assigned to the variable on the left hand side of the assignment statement. A number of assignment operators exist which cause a binary operation to take place between the left hand side and the evaluated right hand side prior to storage of the value (e.g. "x =+ 2;" adds two to "x").

The major control statements in C are "while", "do-while", "for", "switch", "goto", "break" and "continue". The "while" statements causes execution of a group of statements as long as an expression is true. A "do-while" statement is like a "while" except that the control expression is evaluated after the execution of the group of statements. Therefore, the "do-while" statement is always executed at least once. The "for" statement is an extension of the while which provides control variable initialization and loop incrementation. The "switch" statement allows the execution of one of a group of statements labeled as cases based on the value of an expression. The "goto" statement transfers control to a labeled statement in the usual fashion. "Break" and "continue" exist to provide for label-free loop termination and skipping.

A subroutine library is provided for use with C programs. It contains system calls for I/O and other functions. In addition, it contains routines for formated output and for the standard functions of analysis. For a more complete description of C, see Ref. 10 and 18.

III. IMPLEMENTATION OF THE CODASYL DESIGN.

A. Implementation Philosophy.

The overriding consideration in implementing the DBMS was to avoid any modifications or additions to the existing UNIX facilities. This decision was made for a number of reasons. First, other research is being conducted in the Computer Laboratory utilizing the UNIX operating system as a research tool. Running systems which require non-standard veresions of UNIX interfers with the control environment and generally makes other operating system modifications more difficult. Second, a modification to the operating system must be re-applied whenever a new release of UNIX is installed. Third, the chances of the DBMS being transported to other UNIX sites is far greater if it runs under a standard UNIX. Finally, the research goal of determining if the DBMS could be implemented in a variety of environments would be subverted by modifying the operating system environment.

Since the most notable feature in UNIX is the design of its file system, it was decided to utilize the file system whenever possible rather than acquiring a large block of physical media space and letting the DBMS manage it. This philosophy was expected to simplify the problem of mapping data to media and thereby reduce the size and complexity of the DBMS and insulate the DBMS from changes in the hardware.

The final guideline was to implement as large a useful subset of the features in the CODASYL design as feasible under the above assumptions. Creative extensions to the CODASYL design were avoided since these would tend to obscure the research goal of being able to measure the utility of the CODASYL network model against the INGRESS relational model. Efforts were directed instead to the realization of the goals of the CODASYL DDLC, which are very ambitious in themselves.

None of the above assumptions should be taken as precluding the possibility of future modifications to enhance the implementation either of UNIX or of the features of the CODASYL design. The intent of the implementation philosophy described herein was to produce a standard CODASYL DBMS running under a standard UNIX for use as a baseline product.

B. Organization of a Data Base.

Virtually all information about the data base described by a particular schema is contained in a special directory. The only exceptions are certain files which are created for the life of a user process and then discarded. The data base, its schema and its directory all have the same name. Although it is possible for directories with the same name to exist in a UNIX file system, no two data bases should have the same name. The files within the directory associated with a data base (called a schema directory) contain the source and object schemas, the schema Data Base Manager

(DBM) program and all the non-temporary data within the data base. Specific files will be mentioned when they are relevant to the discussion. Appendix B contains a complete listing of the files associated with a schema.

C. Operating Environment.

The environment for both system maintenance and user access of the data base is provided by the DBM Request Processor ("dbm"). This program is a general purpose command language processor used to provide interface with any data base. Appendix C contains a description of the functions of the DBM Request Processor.

When a user wishes to execute a program which accesses a data base, he executes dbm and specifies the appropriate schema name. He then gives dbm an "x" command and specifies a path name to the user program and arguments to be passed to the program. Dbm opens two pipes as interprocess communication channels and forks off two children. Through "exec" calls, these processes become the schema DBM program and the requested user program respectively. Both programs are passed the file descriptors of their respective ends of the interprocess communication pipes as part of their calling arguments. The child destined to become the schema DBM changes directories ("chdir") to the schema directory prior to executing the schema DBM.

Once the user and the schema DBM program are established, dbm waits until they have terminated before accepting any more commands. During execution, the user program and the schema DBM pass requests and data through the two pipes with the user program executing in the user's working directory and the schema DBM executing in the schema directory. It is possible for other users to execute concurrently using the same data base; however, each user has his own version of the schema DBM and a separate set of interprocess pipes.

This operating environment differs from the one envisioned by the CODASYL DDLC in that each user process is interfaced to its own copy of the DBMS routines. Each copy has its own buffers and no knowledge of the existence of other copies, except that which it can derive from the state of files within the schema directory. In contrast, the CODASYL designers described implementation of a single copy of the DBMS routines which would concurrently communicate with all the users and have communal system buffers for servicing all user requests [Ref. 2]. The reasons for this difference are twofold.

First, even though pipes are the only reasonable method for interprocess communication, they are limited in that two processes may communicate only via a pipe opened by a common ancestor. In general, the only common ancestor of processes spawned by different users is UNIX and, although the mechanism exists for finding the process id of a process ("ps"),

no mechanism exists for requesting UNIX to open a pipe to a designated process.

Second, even if a mechanism existed to connect a single copy of the schema DBM to users, that single copy could not muster sufficient resources to service them. In particular, a process may have only fourteen simultaneously open files and each user would require two files open (its pipes) on a dedicated basis. Thus, since access to an area requires two files to be open, a single schema DBM having several users each requiring several areas would develop a thrashing condition in which almost every access to the data base would incur the overhead of two file opens and two file closes. Additionally, a problem with memory buffer contention might arise, although this problem would probably be less critical.

The existence of separate copies of the schema DBM does not mean that the program must be duplicated in memory for each of its current users. UNIX provides a facility for processes executing the same program to share the same text segment, thus only the data and stack segments are replicated for each process. The consequences of the multiple schema DBM environment will be discussed later.

D. Source and Object Schemas.

When a new data base is to be created, a source schema description must be prepared. A schema directory should be created (using the UNIX function "mkdir") to contain the source language version of the schema. The source schema description must reside in a file whose name is formed by prefixing the schema name with "s." and which is located in the schema directory. The UNIX text editor ("ed") is suitable for entering the source schema description. The source schema is coded in a modified form of the CODASYL DDL described in Ref. 2. Differences between the DDL of Ref. 2 and the UNIX DBMS DDL are discussed in Appendix G.

Once the source description of the schema is entered, it must be compiled into an object version. This compilation is accomplished via the "c" command of the DBM Request Processor. The object version of the schema consists of two files which contain the schema DBM program and the encoded schema description, respectively. The schema DBM interprets and services all user requests for access to the data base, while the encoded schema description is a compact symbolic form of the schema's structure. The name of the file containing the schema DBM is the schema name prefixed by "dbm.". The schema DBM is discussed in Section III.L below. The encoded schema description file is used to initialize the schema DBM program and for information about the data base during the move and garbage collection functions of the DBM Request Processor. Appendix D contains a description of

the format of the schema description file.

E. Interprocess Communication.

The schema DBM and the user process communicate via the pipes set up for them by dbm. These pipes may be read and written just as if they were ordinary open files. Messages of a predefined format are sent and received by both processes. The first message sent is the initial call message from the user process. This message is triggered by the C DML "permit" function and contains an encoded description of the sub-schema. The schema DBM response to the initial call includes the index numbers for all the entities and attributes contained in the sub-schema description. Subsequent user program messages are requests for data retrieval or update and are made utilizing the index numbers acquired in the initial call. Since the schema DBM will receive an end of file condition when trying to read the interprocess channel after user termination, no indication need be given to the schema DBM that the user program has terminated.

Messages sent by the schema DBM fall into two categories: normal responses and error messages. The error codes in error messages correspond to those used by the C DML. For a description of the format of all the interprocess messages see Appendix E.

F. Data Base Keys.

In the CODASYL DBMS each record must be identified by a unique value called its data base key. This key is assigned when the record is created and remains with it for the life of the record. The ability to map a record's data base key to the record in a quick and unambiguous fashion must be provided since the key is used for direct access. The key's order relative to all other keys in the area must be well defined since it is used for sequential access. However, the record is allowed to move around in physical media space as long as it stays within the same area. Section III.G below will discuss how the problem of satisfying all the criteria for data base keys was resolved.

The format of a data base key is shown in Fig. 1. This format makes possible 255 areas (area zero is the null area) each containing up to 16,777,215 records.

```
Bits:    31      24 23                     0
         ---------------------------------
Fields:  | area # | record # in area |
         ---------------------------------
```

Data Base Key Format.

Figure 1.

When a record is first created and assigned to an area, that area's index number becomes part of its data base key. It is not possible, therefore, for a record to migrate to

another area. The record number is a purely logical ordering and is implemented as described below.

G. Area Handling.

Each area specified in the schema has associated with it two files. The first of these is the file containing the data stored in the area. Its name is the same as the name of the area. The second file is the data base key file for the area. Its name is the area name prefixed by "k.".

The data base key file is organized into 24-bit entries. Each entry contains the starting byte offset into the area data file of the record associated with a particular data base key for that area. The data base keys are mapped to the entries sequentially. That is, multiplying the record offset portion of the data base key by three yields the starting byte offset of the entry in the key file associated with that data base key. If the value of the key file entry for a data base key is zero then that key is null (i.e. unassigned). The first entry slot in the data base key file for each area is reserved for storing the highest used key in the area to facilitate sequential searching of an area. Thus record number zero is undefined in each area.

Data records are stored in the data file with a three word prefix. The first three bytes of the prefix contain the record number portion of the data base key for that record. The fourth byte contains the record type (a total

of 255 types are possible). The last two bytes of the prefix contain the size in bytes of the record (maximum record size is therefore 32767 bytes).

As records are created and added to the area, they are entered sequentially in the area data file following the last record written (data base keys may be assigned by any algorithm, however). The records will remain in their original locations until they are deleted or moved during garbage collection. If a record is moved to a new location, its key file entry is updated accordingly. Whenever a record is moved or deleted from the area data file, the first two words of the prefix at its former location are zeroed.

The positioning control mechanism provided in the schema DDL is implemented via data base keys. The area control is handled in a trivial fashion since the area index number is a part of the data base key. The positioning of a record "near" another record is accomplished by assigning the record being added the next available data base key following the data base key of the record it is to be "near". This method speeds access to records clustered "near" one another when they are used in conjunction with each another since their data base key file entries are likely to be in the same block. Additionally, the garbage collection function of the DBM Request Processor automatically re-sequences the records in the area data file to be in ascending order of data base key. After garbage collection the records are

clustered in the area data file as well.

If a data base key assignment algorithm causes a sparse key space, the storage needed for recording the key entries is minimized by the fact that UNIX only allocates storage for blocks actually accessed. For example, if a data base key were to be allocated whose key entry block would be 200 blocks beyond the current end of the data base key file, only the block containing that entry would be allocated. Even though the apparent size of the file would have increased by 200 blocks, the intervening 199 blocks would not be assigned any physical media space. Unfortunately, if an empty block is read, space for it is allocated. This means that if the area in the above example were ever scanned sequentially, all the non-allocated blocks in the data base key file would be allocated.

Due to the deletion and addition algorithms, gaps will develop in the data file during the course of processing. The total size (in bytes) of these gaps is maintained in the first four bytes of the area data file. During the execution of the schema DBM, the amount of wasted space is accumulated and at the end of the run the area data file waste count is incremented. Since this method permits more than two billion waste bytes to be accumulated, no provision is made for overflowing the waste count. When the waste count gets to an unacceptable size, the DBM Request Processor can be used to effect garbage collection.

Areas which are designated as temporary areas are handled in a slightly different manner. Since a temporary area is local to the user process opening it, the file names for such an area are suffixed with the process id of the user process. Since the process id uniquely identifies the process, these names uniquely identify a particular version of a temporary area. Additionally, the files associated with a temporary area are allocated in the "tmp" directory and are deleted when the process is terminated. The "tmp" directory has the characteristic that if a system crash occurs, the files within it are lost.

The files associated with any area are automatically created by the schema DBM if it attempts to open them and they do not exist. This means that when a schema is first created, its areas will come into being automatically as soon as they are needed.

H. Access Methods.

There are five access methods which may be used for locating a record in the data base: direct, sequential, calculated, chained and indexed.

1. Direct Access.

If the data base key of a record is known, it may be accessed directly using the data base key mapping mechanism described above. Every access to the data base ultimately involves direct access once the data base key is known.

Unadorned direct access is provided to the user through record currency and through explicit key record selection expressions (see Section IV below).

2. Sequential Area Scan.

The "next" and "prior" records in an area are accessed through a sequential scan. The algorithm successively increments or decrements the record number in the current data base key until the next or previous non-null data base key is found.

3. Calculated or Hashed Access.

A data base key may be developed by a hashing algorithm which uses data in a record for a hash key. The schema record entry for records accessed by hashing must have a location mode of "CALC". During record creation, CALC key collisions are resolved by a forward linear scan until a null key is developed. A key link is established in the synonym record to enable future access to the new record. If multiple collisions occur on the same data base key, a linked list is developed leading to the last synonym added. A standard schema DBM utility routine is used for hashing ("randkey"). If non-standard hashing for a record is desired, a data base procedure may be specified in the location mode clause of the record's schema entry. When a record of a type using a non-standard hashing procedure is to be added, the data base procedure declared in the location mode clause for the record will be called to provide

the data base key.

4. Chained Access.

The default method of set linkage is via chaining. The links in a chain consist of data base keys stored in the records to be linked. The owner record of a set contains links to the first and last member records in the set. Each member record contains a link to the next record in the set. If a set is defined as "PRIOR PROCESSABLE" in the schema, each member record has a link to the previous member of the set. A member record defined as "LINKED TO OWNER" in the schema will contain a link to the owner record of the set. The link-to-next-record in the last record of a set and the link-to-previous-record in the first record of a set both point to the owner record of the set.

5. Indexed Access.

Sets which are singular or dynamic have indices as their primary access method. Additionally, indices are used for secondary set linkage to implement "SEARCH" keys defined in the schema for a record type. An index consists of a list of data base keys ordered as specified in an "ORDER" or "SEARCH" clause. When record selection is through set membership with data field values specified (see Appendix A, Section B.1.a), an appropriate index will be used to gain access to the record. See Section III.I below for a further discussion of indices.

I. The Schema Index File.

All indices created in the data base are stored in the schema index file. The name of this file is the schema name prefixed by "index.". Each index in the file is organized into 512 byte blocks each of which has the format shown in Fig. 2.

```
struct iblock{
    int iblink;         // link to previous index block
    int iflink;         // link to next index block
    char ientry[508];  // up to 127 index entries
    }
```

Format of an Index Block.

Figure 2.

When the backward link field (iblink) in the first block of an index is zero, the index is not in use. Minus one indicates that it is in use. The forward link field (iflink) of the last block in the index is zero. The entry array (ientry) contains up to 127, four byte data base keys. Null entries at the end of an index block are all zeroes. When an index is first created, seven empty entry slots are left at the end of each block for future growth.

An index is searched by a modified form of binary search. When a record is to be located via the index, the first index block is retrieved and the records corresponding to the first and last data base keys entered in the block are examined. If these records bracket the desired record,

a binary search is conducted through the index block to find the desired record. If the desired record is not associated with the index block, subsequent index blocks are read and the last record for each block is examined to determine if it brackets the desired record. When the correct block is found, a binary search of that block is used to find the record. For an index with k blocks having an average of n entries in each block, the average number of records examined in locating a record is approximately (k / 2) + m, where m is the log base two of n.

When data base keys are added to the last block of an index, a new last index block is created whenever six or fewer empty slots remain in the block. If a block other than the last block overflows while a data base key is being added, a new block is inserted into the sequence of blocks. The last seven entries of the old block are copied into the new block and the new data base key entry is added. Whenever the last key remaining in a block is deleted, the block is removed from the index and freed.

Indices are used for set linkage as well as to facilitate the maintenance of a "NO DUPLICATES" clause refering to items not linked in another way. Indices are linked to the records when needed by storing the starting block number of the index in the record.

In the absence of P and V operators [Ref. 22], an index may be reserved for use by a particular copy of the schema

DBM in the following tortuous manner. When an index is to be accessed, the schema DBM attempts to create a file ("indexdum") in the schema directory. This file is created with a mode that does not allow writing, therefore should another process attempt to create "indexdum" while it is open, the attempt will fail. If a create fails, it is repeated until successful. Once the "indexdum" file has been created, the first block of the index is read from the index file. If the backward link field of the first block is minus one, "indexdum" is closed and the above process is repeated until the backward link is zero. The backward link is then set to minus one and the block written. The "indexdum" file is then destroyed. If an index is not available, the schema DBM releases any indices allocated in order to avoid deadlocks.

When an index is to be released, the backward link of the first index block is set to zero and the block is rewritten. Thus, only one schema DBM can use a particular index at any given time. This system avoids integrity problems stemming from simultaneous update of an index block by two different copies of the schema DBM.

During the course of day-to-day operations, indices will be created as well as discarded. When an index is of no further use, its blocks must be made available for recylcing. The free blocks thus created, are accounted for on a free block list similar to the free list in a UNIX super block. Block zero of the schema index file is used to

contain the free list. The first word of block zero contains the block number of a free list block or zero if none exist. The remaining 255 words are used to store the block numbers of free blocks.

The free list is maintained as follows. When a block must be added to the free list, its block number is stored in the first available slot in the free block. If all the slots are taken, block zero is copied into the free block; the block number of this free block is stored in the first word of block zero; and the remainder of block zero is filled with zeroes. If a free block must be allocated to an index, the last non-null block number is extracted from block zero and the slot is cleared. If no free blocks are on the list and word zero contains a block number, that block is allocated to the index after first copying its contents into block zero. If block zero is all zeroes, a new block is added to the end of the file for use in the index. Other schema DBM processes are locked out during block aquisition and freeing by the same mechanism used to gain control of an index.

J. Privacy.

The CODASYL DBMS design allows for privacy locks to be established at all levels. It allows separate privacy locks for each function on a resource. Additionally, it allows a privacy lock to be either a string to be matched or a lock procedure. The UNIX implementation features all these

options. Their implementation is accomplished as follows.

When the files in the schema directory are created, UNIX establishes the installation's Data Base Administrator as their owner. By making the access privileges of a file read and write for owner only (UNIX function "chmod"), the Data Base Administrator can prohibit all other users in the system from opening the file. Thus, only the Data Base Administrator (or super user) can directly read or write a schema file.

The DBM Request Processor may be used for both system maintenance and to initiate user execution. However, since the DBM Request Processor does not set-user-id, it can only perform system maintenance functions when used by the Data Base Administrator.

Since the schema DBM program file is executable by any user, the DBM Request Processor can initiate it to process user requests. Since the schema DBM does a set-user-id to the user id of the Data Base Administrator, it can access the schema files as required. A user must not be able to penetrate the schema DBM to gain access to information for which he does not have the privacy keys.

The schema DBM prevents unauthorized access using the following procedure. The schema DBM has a privacy flag for every function/resource pair for which a privacy lock can be defined. When the schema DBM validates the user's sub-schema (contained in the initial call message), it checks

the privacy keys defined in the sub-schema. Each privacy flag for which no lock is defined or for which the sub-schema privacy key is valid, is set to allow access, otherwise it is set to deny access. The user receives no immediate indication as to whether or not his privacy keys fit the locks. If he later tries to access some data base resource in a way for which he did not furnish acceptable privacy keys, his request fails. Once an initial response message is accepted by the schema DBM, no further unlocking of resources can be done. Thus, in order to access the denied resource, the user program must be terminated and restarted using a fresh copy of the schema DBM which must be provided the proper privacy keys. Thus, no single execution of a user program can, through trial and error, determine the valid privacy keys.

K. Integrity.

As previously mentioned, the CODASYL DDLC envisioned that the DBMS routines would be contained in a single process which would service all users. That concept guarantees the integrity of the data base since simultaneous update of the data base is impossible. Additionally, a concept called "keep" status is included in the COBOL DML [Ref. 3]. A record has automatic "keep" status for a run-unit during the time it is the current record of that run-unit. A run-unit can also request "keep" status for a record if it desires to be informed of what happens to the record. If a run-unit

modifies or deletes a record which has "keep" status for another run-unit, the run-unit having the record in "keep" status will be notified of the action. Although the "keep" mechanism does not resolve the problem of concurrent update, it does provide a mechanism for identifying potential problems. "Keep" status allows run-units to update the data base while still allowing access to it.

Since each user process is coupled to its own version of the schema DBM, none of the above features can be readily implemented. As a consequence, if multiple schema DBM programs concurrently open an area for update, data base integrity problems are virtually assured. If, however, all users open any area to be updated for protected or exclusive use (see Appendix A, Section B.2.b), no integrity problems can arise.

An area opened for protected use cannot be opened by another process for update. An area opened for exclusive use cannot be opened at all by another process. The mechanism for insuring that these rules are enforced is the Logical Usage Block File. This file resides in the "tmp" directory and has the same name as the schema (hence the rule that no two data bases may have same name). It contains the logical usage block which records the opening mode of every area currently open by any copy of the schema DBM. When a schema DBM desires to open an area, it reads the logical usage block and determines whether or not a conflict exists between the opening mode it desires and the modes in use by

other processes having the area open. If no conflict exists, it opens the area and updates the logical usage block accordingly; otherwise, it notifies the user that the open has failed.

In order to avoid problems with simultaneous update of the logical usage block by different processes, a lock out file mechanism similar to "indexdum" is employed. This file is named "opendum" and resides in the schema directory. The "opendum" file is created to lock out other processes while the logical usage block is being accessed.

L. The Schema DBM.

As mentioned earlier, the DBMS compiler must produce a schema DBM program when a schema is compiled. The schema DBM program is composed of two parts: the schema constants and the DBM skeleton.

1. Schema Constants.

The DBMS compiler must produce a C coded temporary file which contains all the schema unique constants neccessary to tailor the DBM skeleton to the schema being compiled. These constants cause the various buffers and arrays used by the DBM program to be allocated sufficient memory to handle the schema. For a description of the constants and arrays involved see Appendix H.

Additionally, this temporary file must include the initialization for the arrays "procpoint" and "procname". These arrays contain, respectively, pointers to and the names of all the data base procedures in the schema. Whenever a data base procedure name is encountered during the initialization phase of the schema DBM, "procname" is searched until the matching name is found and a pointer to the data base procedure is extracted from the corresponding "procpoint" entry.

When the constant file has been generated, the DBMS compiler can use the C compiler to form the schema DBM from the constant file and skeleton DBM. Since pointers to the data base procedures are used as initializing constants in "procpoint", all the data base procedures will automatically be loaded into the output object module after compilation. Both the DBM skeleton and the data base procedures must exist as object modules available to the C compiler. All the external arrays which are dimensioned in the constant file are declared but not explicitly dimensioned in the skeleton DBM. The finished product of the C compiler is an executable schema DBM.

2. The DBM Skeleton.

The DBM skeleton is an object module which contains all the DBMS routines (except data base procedures) required to provide user services for the data base. Appendix F contains a complete description of the DBM skeleton. The pro-

cessing of the skeleton (and thus the schema dbm) is divided into two phases: initialization and user request processing.

a. Initialization Phase.

When the schema DBM is called, it has no information about the organization of the schema or sub-schema. Although all its buffers are the right size and all the necessary data base procedure are compiled into it, it has no knowledge of data base names, privacy locks, set relationships or any other data peculiar to the schema. In order to function, it must read in all the data in the schema description file. Concurrently, it processes the user program's sub-schema which is passed in the user program's initial call message. By validating the sub-schema while initializing the schema, the schema DBM can immediately translate all references into terms of its internal index numbers rather than store data base names. This avoids much of the matching overhead for each user request. If the sub-schema fails the validation, the schema DBM sends the user program an error message, returns to the beginning of the schema description file and restarts initialization. The initialization phase will thus be terminated only if the user program either submits a valid sub-schema or terminates.

b. User Request Servicing Phase.

After a successful initialization, the user request servicing phase begins. This phase consists of a

loop which reads a user message, processes it, and returns a response to the user program. The loop runs until user program termination, at which point the schema DBM terminates.

Processing a user message is accomplished by selecting a service routine based on the message type. One service routine exists for each message type except the initial call, with an additional routine to process invalid message types. During this phase, an initial call is considered an invalid message. Each service routine uses one or more utility routines. Utility routines are general purpose data base access and maintenance primitives which may be used by several service routines.

IV. DESIGN OF THE C DDL AND DML.

A. Design Goals and Decisions.

The augmentation to the C language was designed to provide a natural interface between the C language and the DBMS without reducing its ability to support a COBOL DDL and DML. Accordingly, the C DML was designed to have as much functional similarity to the COBOL DML as was feasible. This goal was adopted to support the research objective of testing CODASYL's contention that the DBMS could support a variety of sub-schema DDL's and DML's. See Section C below for a comparison of the COBOL and C DDL and DML.

One of the desirable goals of a DBMS is to provide program independence from the definition of the data base. Additionally, one of the primary design philosophies of C was economy of expression. In order to facilitate both of these goals, the C DDL provides for describing only a minimal subset of the relationships and restrictions which appear within a schema description. The DDL is restricted to describing the names and privacy keys for areas, records, data items, data aggregates and sets; and the membership of record types in set types. Data types of items and aggregates must be specified as well, but this information may be different from the data types recorded in the schema description. Additionally, the DDL need only describe those

portions of the schema the program is interested in manipulating. Since this information is the only data absolutely necessary to the DML, unless major changes affecting the validity of the program logic occur in the schema, a recompilation of the program should seldom be needed. The programmer obviously needs to know a lot more about the schema, however he can (and should) obtain this information from an installation Data Element Dictionary [Ref. 19].

A fourth goal was to integrate the C DDL and DML into the host language's structure whenever possible. Accordingly, the DDL was grouped into a special external function and the DML functions have been given formats similar to other C special functions such as "return". DML logical expressions are compatible with normal C expressions.

B. Major Concepts.

This section describes some of the concepts essential to the implementation and use of the C DDL and DML. For a complete definition of the C DDL and DML, see Appendix A.

1. Currency.

The concept of currency is central to the navigating of access paths in the DML. The user process as well as each record, set and area type known to it have a current record associated with them. This currency is established by the execution of a "find". When a record is found, it becomes the current record of the process, its record type,

the area in which it resides and the set type of every set occurence the record participates in as a member or owner. The current set occurence of each set type is the set in which the current record of that set type participates.

Information about a record, including data values if the record was fetched by "get", continues to be available until the record is replaced as the current record everywhere its currency was originally established. For example, if a record is the current record of a particular area, it will remain available as the current record of that area until a "find" is executed which selects a different record residing in the same area.

2. Find versus Get.

The difference between the "find" and "get" functions is that the former locates records in the data base while the latter extracts data values from the data base. When a "find" is executed, the DBMS spans the access paths specified in the record selection criterion and returns all the information about a record necessary to make it current in the appropriate places. For a description of the record selection options available, see Appendix A, Section B.1.b. The information necessary to establish currency includes the selected record's data base key, record type and the set types for the sets it participates in as a member or owner. The area the record resides in can be derived from its data base key.

A "get" is used to access the data values associated with a record. The record must have been made the current record of the process prior to executing the "get". The values of the record's data items are available via pointers associated with each entity for which the record is current. Whether an implementor elects to provide separate buffers for each currency type or merely reassign the value of pointers is immaterial.

3. Independence of Schema and Sub-schema.

A user program can be compiled without reference to the schema description. A program using a sub-schema will continue to execute until the data base name, privacy locks or set memberships described in the sub-schema are changed or deleted in the schema. However, changing entries such as record location modes and set selection clauses in the schema may alter the logic of a program.

The data types of the sub-schema may differ from those of the schema. The DBMS will automatically convert data to the types desired by the sub-schema before delivery. Some type differences, however, may cause an error if the data involved is incompatable. For example, converting a string of characters to an integer will fail if the string contains any non-numeric characters.

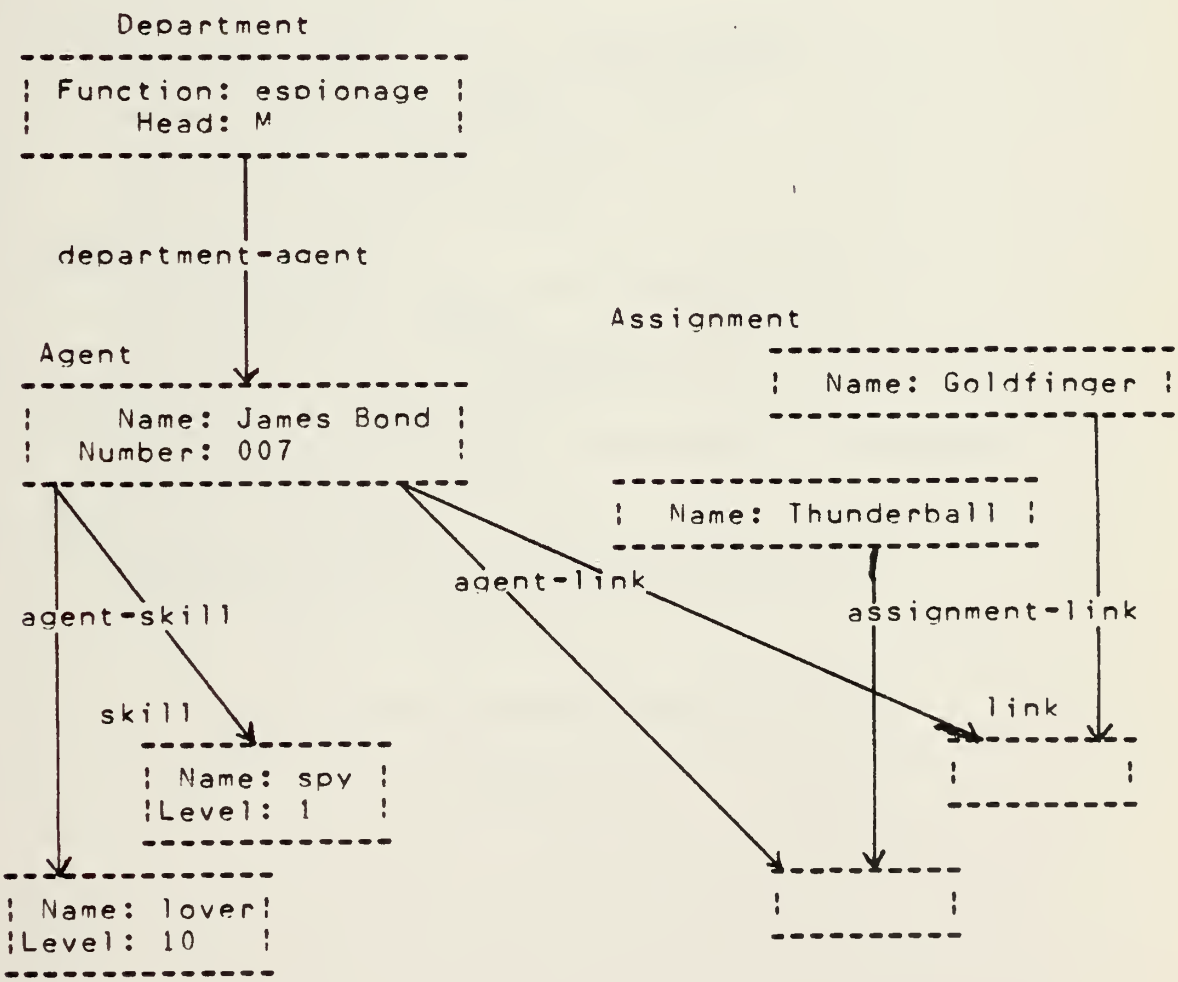
C. Comparison with the COBOL DDL and DML.

The comparision will be made by means of an illustrative example. For a detailed description of the C and COBOL DDL and DML see Appendix A and Ref. 3 respectively. The COBOL portion of the example is patterned after Ref. 20. Several modifications were made to reflect both recent changes in the definition of the CODASYL DBMS and UNIX implementation data types.

The example concerns the representation of a personnel data base. Figure 3 is a diagram of the network and records involved. In Fig. 3, the rectangles represent records and the arrows point from set owners to set members. Record names are written above rectangles, item values within. Set names are superimposed over the set linkage arrows. Multiple agents may be linked to an assignment and multiple assignments to an agent. The fact that a particular agent is assigned to a particular assignment is represented by the existence of a LINK record which has membership in both the AGENT-LINK set owned by that agent's AGENT record and the ASSIGNMENT-LINK set owned by that assignment's ASSIGNMENT record. The link records are made necessary by the restriction that a record may only be a member of one set of a given type.

Figures 4 and 5 show the DDL description of the schema. The COBOL sub-schema could be essentially an exact copy of the schema. The C sub-schema is shown in Fig. 6. Note that

an agent's number is represented as a numeric character string in the schema and as an integer in the C sub-schema. In the C sub-schema, all the data base names are spelled with lower case letters and with hyphen ("-") replaced by underscore ("←"). The DBMS translates identifiers to allow for this difference in spelling conventions. In the UNIX implementation, data base names spelled with any combinations of upper and lower case letters will be properly recognized.



Network Representation of a Data Base.

Figure 3.

```
    SCHEMA NAME IS PERSONNEL-FILE.

    AREA NAME IS DEPARTMENT-AREA.
    AREA NAME IS ASSIGNMENT-AREA.

    RECORD NAME IS DEPARTMENT.
        LOCATION MODE IS CALC USING FUNCTION
            DUPLICATES ARE NOT ALLOWED;
        WITHIN DEPARTMENT-AREA.
01      FUNCTION; PICTURE IS "A(20)".
01      HEAD; PICTURE IS "A(1)".
    RECORD NAME IS AGENT;
        LOCATION MODE IS CALC USING NUMBER;
            DUPLICATES ARE NOT ALLOWED;
        WITHIN DEPARTMENT-AREA.
01      FIRST-NAME; PICTURE "A(10)".
01      LAST-NAME; PICTURE "A(10)".
01      NUMBER; PICTURE "9(3)".
    RECORD NAME IS SKILL;
        LOCATION MODE IS VIA AGENT-SKILL SET;
        WITHIN AREA OF OWNER.
01      NAME; PICTURE IS "A(20)".
01      LEVEL; TYPE IS FIXED DECIMAL.
    RECORD NAME IS ASSIGNMENT;
        LOCATION MODE IS CALC USING NAME OF ASSIGNMENT;
        WITHIN AREA OF OWNER.
01      NAME; PICTURE IS "A(20)".
    RECORD NAME IS LINK;
        LOCATION MODE IS VIA AGENT-LINK SET;
        WITHIN AREA OF OWNER.
```

Schema DDL Area and Record Entries

Figure 4.

```
SET NAME IS DEPARTMENT-AGENT;
    OWNER IS DEPARTMENT;
    ORDER IS PERMANENT INSERTION IS
        SORTED BY DEFINED KEYS;
    MEMBER IS AGENT MANDATORY
        AUTOMATIC LINKED TO OWNER;
        KEY IS ASCENDING NUMBER;
        SET SELECTION IS THRU DEPARTMENT-AGENT OWNER
            IDENTIFIED BY CURRENT OF SET.
SET NAME IS AGENT-SKILL;
    OWNER IS AGENT;
    ORDER IS PERMANENT INSERTION IS SORTED BY DEFINED KEYS;
    MEMBER IS SKILL MANDATORY AUTOMATIC;
        KEY IS DESCENDING LEVEL;
        SET SELECTION IS THRU AGENT SKILL OWNER
            IDENTIFIED BY CURRENT OF SET.
SET NAME IS AGENT-LINK;
    OWNER IS AGENT;
    ORDER IS PERMANENT INSERTION IS IMMATERIAL;
    MEMBER IS LINK MANDATORY AUTOMATIC LINKED TO OWNER;
    SET SELECTION IS THRU AGENT-LINK OWNER IDENTIFIED
        BY CALC-KEY EQUAL TO CURRENT-AGENT.
SET NAME IS ASSIGNMENT-LINK;
    OWNER IS ASSIGNMENT;
    ORDER IS PERMANENT INSERTION IS IMMATERIAL;
    MEMBER IS LINK MANDATORY AUTOMATIC LINKED TO OWNER;
        SET SELECTION IS THRU ASSIGNMENT-LINK OWNER
            IDENTIFIED BY CALC-KEY EQUAL TO
                CURRENT-ASSIGNMENT;
```

Schema DDL Set Entries.

Figure 5.

```
ddl{
    schema personnel←file;
    area department←area;
    area assignment←area;
    record department{
        char function[20];
        char head[1];
        }
    record agent{
        char first←name[10];
        char last←name[10];
        int number;
        }
    record skill{
        char name[20];
        int level;
        }
    record assignment{
        char name[20];
        }
    record link{}
    }
    set department←agent owner is department{
        member agent;
        }
    set agent←skill owner is agent{
        member skill;
        }
    set agent←link owner is agent{
        member link;
        }
    set assignment←link owner is assignment{
        member link;
        }
    }
```

C Sub-schema Entries.

Figure 6.

1. Query 1.

The first query is designed to extract the skills of agent 007. The procedure is to initialize the agent number, FIND the agent and print the agent number, skill name and skill level for each skill the agent has (if any). The COBOL and C realizations of Query 1 are shown in Fig. 7 and 8 respectively. In both versions access to the agent is via the CALC key in the AGENT record and the appropriate AGENT-SKILL set is automatically selected when the agent 007 becomes current. The agent record need not be fetched since none of its data fields are needed.

```
        OPEN ALL.
    FIND-AGENT-RECORD.
        MOVE '007' TO NUMBER OF AGENT.
        FIND AGENT RECORD.
    READ-FIRST-SKILL.
        FIND FIRST SKILL RECORD OF AGENT-SKILL SET.
        IF ERROR-STATUS = 0326 GO TO ALL-DONE.
    PRINT-SKILL.
        GET.
        DISPLAY 'AGENT = ', NUMBER OF AGENT, ', SKILL = ',
            NAME OF SKILL, ', LEVEL = ', LEVEL OF SKILL.
    READ-NEXT-SKILL.
        FIND NEXT SKILL RECORD OF AGENT-SKILL SET.
        IF ERROR-STATUS = 0307 GO TO ALL-DONE,
        ELSE GO TO PRINT-SKILL.
    ALL-DONE.
```

Query 1 Coded in COBOL.

Figure 7.

```
dbopen();
agent.number = 7;
    find(agent);
    for(find(first skill of agent←skill);!error.status;){
        get();
        printf("Agent = %s, Skill = %s, Level = %s\n",
            agent.number,skill.name,skill.level);
        find(next skill of agent←skill);
        }
```

Query 1 Coded in C.

Figure 8.

2. Query 2.

The second example query is designed to find all department heads concerned with the assignment "THUNDERBALL". The procedure is as follows. FIND the ASSIGNMENT record whose NAME is "THUNDERBALL". For each LINK record in the assignment's ASSIGNMENT-LINK set,

. find the link's owner in the AGENT-LINK set it belongs to,

. find that AGENT record's owner in the DEPARTMENT-AGENT set it belongs to and

. print the assignment name and department head.

The COBOL and C realizations of Query 2 are shown in Fig. 9 and 10 respectively. In both versions, once the desired link is made current, the appropriate DEPARTMENT record can be reached via the AGENT-LINK and DEPARTMENT-AGENT set. Note that only the DEPARTMENT records need to be fetched since all the spanning of access paths is accomplished through currency information.

```
    OPEN ALL.
FIND-ASSIGNMENT-RECORD.
    MOVE 'THUNDERBALL' TO NAME OF ASSIGNMENT.
    FIND ASSIGNMENT RECORD.
FIND-FIRST-LINK.
    FIND FIRST LINK RECORD OF ASSIGNMENT-LINK SET.
    IF ERROR-STATUS = 0326 GO TO ALL-DONE.
FIND-AGENT-LINK-OWNER.
    FIND OWNER RECORD OF AGENT-LINK SET.
READ-DEPARTMENT-RECORD.
    FIND OWNER RECORD OF DEPARTMENT-AGENT SET.
    GET.
PRINT-DEPARTMENT-HEAD.
    DISPLAY 'ASSIGNMENT = ', NAME OF ASSIGNMENT,
        ', HEAD = ', HEAD OF DEPARTMENT.
FIND-NEXT-LINK.
    FIND NEXT LINK RECORD OF ASSIGNMENT LINK SET.
    IF ERROR-STATUS = 0307 GO TO ALL-DONE
        ELSE GO TO FIND-AGENT-LINK-OWNER.
ALL-DONE.
```

Query 2 Coded in COBOL.

Figure 9.

```
dbopen();
for(i=0;i<12;i++)assignment.name[i]="THUNDERBALL"[i];
find(assignment);
for(find(first link of assignment←link);!error.status;){
    find(owner of agent←link);
    find(owner of department←agent);
    get();
    printf("Assignment = %s, Head = %s\n",
        assignment.name,department.head);
    find(next link of assignment←link);
    }
```

Query 2 Coded in C.

Figure 10.

V. CONCLUSIONS AND RECOMENDATIONS.

A. Conclusions.

Since the DBMS compiler and the C language augmentation are not yet implemented, it is difficult to fully evaluate the effectiveness and efficiency of the DBMS. In general, it can be said that the UNIX file system seems to be a very hospitable environment for developing a DBMS, however the operating system facilities of UNIX are not nearly as well suited to supporting this development. The DBMS is measured against some of the goals of DBMS as they are presented in Section II.A.4.

1. Concurrent Retrieval and Update.

The DBMS cannot provide the ability to perform concurrent update of the same area by two users. Although the ability to open an area for unprotected update exists, its use can be disasterous. Concurrency between update and retrieval in an area causes no intregrity problems; however, the user doing retrievals has no way of knowing if the records he is accessing are being modified.

2. A Variety of Search Strategies.

The DBMS supports every form of access path specified by the CODASYL DDLC. These forms are direct, hashed,

sequential and indexed.

3. Centralized Placement Control.

Placement control by the DBMS is a purely logical mapping with the UNIX file system providing centralized placement control for the data onto physical media.

4. Device Independence.

Device independence is almost total for any file in UNIX. The DBMS (and therefore the user program) is unaware of either the types or number of devices in the system.

5. Privacy of Data.

The complete privacy mechanism in the CODASYL design has been implemented. The DBMS itself should be relatively secure. A program could be written to call the schema DBM repeatedly and determine a privacy key by trial and error, but using data base procedure privacy locks which notify a security console or terminate the program when a violation occurs can greatly reduce the effectiveness of trial and error "lock picking".

UNIX itself, however, is too easily penetrated [Ref. 21]. Locating and plugging all the holes in UNIX may be impossible.

6. Independence of Schema and Sub-schema.

The DBMS provides the maximum amount of independence possible under the CODASYL design. In fact, user programs could be compiled without any reference at all to the schema.

B. Recomendations.

1. Enhancement for Concurrent Update.

In order to enhance the ability for concurrent use of a data base, the following approaches might be taken.

a. Centralized Schema DBM.

UNIX could be modified to provide a mechanism for establishing interprocess communication to any designated process. This would enable implementation of a centralized schema DBM as the CODASYL DDLC intended. This alternative remains impractical for the reasons Section III.c, i.e., the schema DBM would run out of file resources. Additionally, the UNIX modification would have an unknown but probably major impact on the operating system's design.

b. System P and V Call.

UNIX could be modified to provide a system call for P and V operators. For a discussion of P and V operators see Ref. 22. If a fast P and V facility were available, a schema DBM could temporarily halt all update or

access to an area while performing modifications. "Keep" status could be implemented, if desired, by storing indicators in the record itself. The impact of such system calls on UNIX's design philosophy is expected to be minimal.

Additionally, existing communications between schema DBM programs could be speeded up. Specifically, the methods used with "opendum" and "indexdum" to lock out simultaneous update are essentially a "test and set" operation which could be implemented more efficiently with P and V system calls.

2. Enhancement for Faster Access.

In the absence of usage data, it is difficult to estimate the access response speed of the DBMS. However, a logical extension to the access methods provided by the DBMS would be multilevel indices. The index structure now in the DBMS is essentially an index sequential access scheme which could be upgraded to the multilevel structure which is typical of such indices. For a disscusion of multilevel index sequential access method, see Ref. 13. Use of a two leveled index would divide the average number of records scanned to find the right index block by the average number of entries in the index blocks.

3. Automatic Garbage Collection.

Since only the Data Administrator can initiate garbage collection, the wasted space growth rate in a data base

may become a problem. Some consideration should be give to having the schema DBM automatically garbage collect when the waste in an area reaches a critical level. This thesis did not address the problem of automatic garbage collection due to the difficulty of determining what amount of wasted space is critical.

APPENDIX A. C LANGUAGE DDL AND DML.

A. C Language DDL.

The DDL in C is designed to interface the subschema description with the schema description with a minimal requirment for path information from the user and maximal similarity with existing C language constructs. In the following discussion, words enclosed in apostrophes denote variable data. When a 'lock' is specified, the data item must be of type "character pointer". All 'db identifiers' specified must match the appropriate data base names in the schema after translation of lower case into upper case and underscore into dash. All DDL statements are enclosed in a "ddl" routine with the following format:

"ddl{...ddl statements...}".

The ddl routine should appear prior to any DML statements. It may be contained in a file INCLUDE'd at an appropriate point in the program. The statements in order of appearance are as follows.

1. Schema Entry.

The "schema" statement identifies the schema name and its privacy lock. Its format is

"schema 'db identifier' with lock 'lock';".

The 'db identifier' must match the schema name and the lock

must match the privacy lock for the schema entry (see Ref. 2, section 3.1.0).

B. Area Entries.

For each area to be used, an area entry must be made. These entries must be in the same order as the area entries in the schema. The format of an area entry is

"area 'db identifier list' lock is 'lock list';",

where a 'db identifer list' is one or more comma separated 'db identifier's. A lock list is one or more comma separated lock entries of the form

"'lock' for 'modifier' 'function'",

where the modifier is optional. For an area entry, the allowable modifiers are "exclusive" and "protected" and the allowable fuctions are "update" and "retrieval". The 'db identifier's and 'lock's must match the area names and privacy locks in the schema area entries (see Ref. 2, section 3.2.0).

1. Record Entries.

Record entries must be made in the same order as the corresponding record entries in the schema. A record entry is similar in construction to a C language structure definition. Its format is

"record 'db identifier' lock is 'lock list' {'item list'};",

where the "lock is 'lock list'" phrase is optional. The 'lock list' for records has no modifiers defined. The func-

tions allowed are "insert", "remove", "store", "delete", "modify" and "find". The 'db identifier' and 'lock's must correspond to those of the schema record entry (see Ref. 2, section 4.2.3).

The item list in a record entry is composed of a series of item entries of the following form:

"'type specfier' 'db identifier' ['constant expression'] lock is 'lock list';",

where the "['constant expression']" and "lock is 'lock list'" phrases are optional. A type specifier is of the form "int", "char", "float", "double", "dbkey" or "struct{'item list'}". These data types are identical to those of the C language with the addition of "dbkey". An item of type "dbkey" appears to be an array of four characters to the C user. The 'lock list' for items has no modifiers defined. The permissible functions are "store", "get" and "modify". The item entries may appear in any order in the item entry list with the following restrictions. Items must appear with the same records as in the schema. The data type of the item must be compatable with the schema item. Items of type "struct" must correspond to repeating groups in the schema and have the same dimensionality as in the schema. Items appearing in a repeating group in the schema must appear in the item list of the structure corresponding to that repeating group. Any item in the schema record description may be omitted.

The record names can be used in non-DML statements as structures whose format is identical to the record entry. These record structures are global names and contain the current record of the respective type.

2. Set Entries.

For each set to be referenced, a set entry of the following format must exist:

"set 'db identifier' lock is 'lock list' owner is 'db identifier' *'identifier' {'member list'};",

where the "lock is 'lock list'" phrase is optional. The 'lock list' for sets has no modifiers. The defined functions are "insert", "remove" and "find". The set name 'db identifier' and 'lock's must match those of corresponding set entry in the schema description and all set entries must be in the same order as in the schema description. The second 'db identifier' must match the owner name of the set. The member list is composed of one or more member entries.

3. Member Entries.

A member entry has the form

"member 'db identifier' *'identifier' lock is 'lock list';",

where the "lock is 'lock list'" phrase is optional. The 'lock list' for members has no modifiers defined and the defined functions are "insert", "remove" and "find". The 'db identifier' must be the name of a record defined in the schema as a member record for the set being described.

The *'identifier's named in the set and member entries become global pointers to the appropriate record structure. These pointers can be used to reference the current owner record and member record, respectively, of the set. In addition the set name 'db identifier' is the name of a character array which holds the current record of the set.

C. C Language DML.

The DML has several global names and functions associated with it. Besides the record, item and set names from the ddl routine, there are the pointers "areaname", "recname" and the structure "error". The "areaname" pointer contains the address of the area array containing the current record of the process. The "recname" pointer contains the address of the record structure for the current record of the process. Note the "recname" provides the user with the record type of the current record of the process; but the current record of the process may not be the current record of that type and therefore the record pointed to may not be the current record of the process. The current record of the process will always be available in the area array pointed to by "areaname". "Areaname" and "recname" are set whenever a find or store function is executed. The

"error" vector is a structure with the following format:

```
struct {
    int status;
    int type;
    char *set;        // pointer to set array for error
    char *record;     // pointer to record for error
    char *area;       // pointer to area array for error
    int count;
    } error;
```

The error codes for C DML functions are designed to be compatible with error codes defined in Ref. 3 for the COBOL DML. Additionally, a pointer called "areaid" and an array of four characters called "keyname" exist for the store function.

The use of the DML causes certain identifiers to be generated globally, hence these should be treated as reserved words by the user. These reserved words are:

- all area, record and set names

. areaname	. permit	. remove
. recname	. store	. empty
. error	. member	. owner
. dbopen	. current	. duplicate
. dbclose	. areaid	. keyname
. find	. get	. insert
. modify	. key	

1. DML Expressions.

The DML introduces two additional expression types into C. These are DML logical expressions and DML record selection expressions.

a. DML Logical Expressions.

These expressions evaluate to a true/false value and can be used in a manner identical to normal C logical expressions. Their forms are

(1) "'db identifier' empty", where 'db identifier' must appear as a set name in a set entry. The expression evaluates true if and only if the current set of the type specified has no members.

(2) "member of 'db identifier'", where 'db identifier' must be a set name. It evaluates true if and only if the current record of the process is a member of a set of the type specified.

(3) "owner of 'db identifier'", where 'db identifier' must be a set name. It evaluates true if and only if the current record of the process is the owner of a set of the type specified.

b. DML Record Selection Expressions.

These expressions result in a data base key which can be used to find a record. They are evaluated in part within the user program but must be validated by the

schema dbm program. They must appear only in DML function argument lists. The forms possible are as follows.

(1) Explicit Key. The simplest form of record selection expression is by explict key. The format is "'key'". The 'key' must be either an item of type "dbkey" or evaluate to a character pointer. The contents of the 'key' are used as a data base key. This form is useful for accessing records whose keys are known. It can also be used for applying currency which has previously been suppressed (e.g. "find(key(process));" applies all appropriate currency to the current record of the process).

(2) Owner Record. Selection of an owner record has the format "'db identifier' owner of 'key'", where the "of 'key'" phrase is optional. The 'db identifier' is a set name and the 'key' is an explicit key. If the "of 'key'" phrase is not used, the owner record of the current instance of the set specified is selected; otherwise the owner in the set type specified for the record identified by the 'key' is selected.

(3) Relative Selection. This form allows the selection of a particular record from an area or set based on a location criterion. The expression has the format "'criterion' 'db identifier' of 'db identifier'". The first 'db identifier' is optional and is the name of a record type. The second 'db identifier' is the name of an area or set type. The 'criterion' determines the location within

the area or set from which the record will be selected. The allowed criteria are "next", "prior", "first", "last" and an expression which evaluates to an integer. When the record type is included, only occurences of that type record will be considered for selection. The 'criterion' refers to the ordering of the area or set. The ordering of an area is considered to be ascending sequence by data base key. "Next" and "prior" are relative to the current record of the area or set. If the current record of the set is the owner record, "next" and "prior" are equivalent to "first" and "last" respectively.

(4) CALC Key. If a record type is defined in the schema as having a location mode of CALC, the format "duplicate 'db identifier'", where "duplicate" is optional, may be used. The 'db identifier' is a record type defined in the schema to have location mode CALC. Prior to the evaluation of the record selection expression, the items in the record designated as part of the CALC key must have been initialized to the desired values.

If the "duplicate" phrase is included, the current record of the process must be of the specified type and have the same CALC key as in the record buffer. If these conditions are satisfied, a synonym to the current record is selected.

(5) Data Value. Selection by data value is possible using the format "duplicate 'db identifier' via 'set select' 'db identifier' == 'db identifier' ...", where the phrase "== 'db identifier' ...", and the word "duplicate" is optional. The first 'db identifier' is a record name; the second, a set name; and the string of 'db identifier's is made up of items in the named record type. The 'set select' phrase consists of either the word "current" or the format "'db identifier' ... select". The 'db identifier' string in the 'set select' phrase is of made up of the items needed for the selection path specified in the SELECTION clause for the the named record type and named set type.

If the word "duplicate" is omitted, the expression selects the first record occurence in the appropriate set with values matching those of the items in the string of 'db identifier's. If the string is not specified, the first record of the named type in the set is selected. When the list of items is specified, the items in the list must have been initialized to the desired values prior to evaluation of the record selection expression. If "current" is included, the current instance of the named set is used, otherwise the set used is selected on the basis of the selection criteria in the schema for the named record type as a member of the named set type.

If the word "duplicate" is included, "current" must be included and the item list is mandatory.

The record selected will be the next record in the set matching the current record of the process in the fields named in the item name string.

2. DML Routines.

The ability to access, retrieve and update records is provided by DML routines. The routines are divided into area manipulations (dbopen, dbclose), record manipulations (key through delete) and set manipulations (insert, remove). The permit function does not fit into any of these categories. Considerable overlap between categories exists among the other functions. All functions have the same form as normal C subroutine calls. In the following description, all error codes listed have a two decimal digit major code and a two decimal digit minor code specifying the function and specific error respectively.

a. Permit. The permit function must be called only once and must be before any other DML functions. It causes validation of the subschema by the schema DBM program and establishment of the privacy permissions required. If the schema lock is violated, an error code of 0010 is returned in error.status. If any other privacy lock is failed, no indication is given until the user program attempts to use the feature not properly unlocked. If a mismatch occurs between the schema definition and the subschema definition, error code 0060 will be returned in error.status. When this occurs, error.count will contain the number of incompatibil-

ities; error.type will be 1, 2 or 3 depending on whether the first error encountered was in an area, record or set entry; error.area, error.record and error.set will indicate the first erroneous entry in the area, record and set entries respectively. The entry number returned identifies entries in the C subschema and is zero if no errors were encountered. If any other DML function is attempted prior to permit, error code nn61 will be returned, where nn indicates the function attempted.

b. Dbopen. Prior to processing any records in an area, the user program must call dbopen to open the area. Dbopen parameters are an opening mode and a list of area names. The opening mode is an octal code formed as follows. If the low order bit is 1, the mode is for update and retrieval otherwise it is retrieval only. The next most significant two bits are zero for concurrent update permited; 1 for concurrent retrieval but no concurrent update (protected mode); and 2 or 3 for no concurrent use permitted (exclusive mode). All the areas in the parameter list are opened in the specified mode. If no area list is specified, all the areas in the subschema are opened. For all temporary areas opened, a mode of exclusive update is assumed no matter what mode is specified. Modes allowing concurrent processes to update areas are included for compatability purposes; however, unless the implementation of the data base management system is modified, these modes can cause severe integrity problems.

To successfully execute a find, store, delete or close, appropriate areas must be open as follows: all areas which contain any record occurence which would be deleted or removed by a delete statement and all areas which are the objective of a close function. If any of these functions fails to meet these conditions, error status nn01 is returned, where nn indicates the function attempted.

In addition to the areas containing the object records of the functions cited above, there are additional (i.e. implicit) areas which could be impacted by DML functions. This impact can be of two forms: the DBM program requires information contained within the implicit area (in which case the area must be "available") or the DBM program must alter the information contained in records in the implicict area (in which case the area must not only be available, but it must permit the necessary alteration). Implicit areas requiring modification are termed "affected".

A user may assume the following areas will be affected: all areas containing any record which participates in a set occurence into which a record is to be inserted or from which a record is to be removed or deleted and all areas containing any records which participate in any set occurence whose membership or sequence is altered by a store or modify function. If an implicit area which is affected is not open, error code nn21 will be returned to error.status, where nn indicates the function attempted.

To successfully execute insert, remove, store, delete or modify functions, both explicitly and implicitly affected areas involved must be opened for update. If any of the involved areas are open for retrieval only, an error code of nn09 will be returned to error.status where nn indicates the function attempted.

Record occurences which are in the search path of a find or an implicit find which is the result of a store, remove or delete function need only be in areas which are available. In order for an area to be available, it must not be opened for an exclusive mode by a concurrent process. Although it need not be open, the full overhead of a dbopen and dbclose will be incurred for each implicit reference to an area which is not open. If an implicit area is not available, error code nn18 will be returned to error.status, where nn indicates the function attempted.

Any attempt to execute a dbopen function which would result in a usage mode conflict for any area will result in the failure to open every area. Additionally, error code 0929 will be returned in error.status. A usage mode conflict will occur under the following conditions: any mode of update on an area opened in an exclusive or protected mode by another process; any protected mode on an area opened for update by another process; exclusive mode on an area opened for any mode by another process; and any mode on an area opened for exclusive use by another process. In order to prevent deadlock conditions, a process should open

all areas needed for exclusive or protected use in one dbopen. If a dbopen fails because of usage conflict, the process should close any other open areas obtained previously.

If a privacy lock is violated, error code 0910 is returned in error.status. If an area opened was already open, warning error code 0928 is returned in error.status. The total number of errors encountered is returned in error.count.

c. Dbclose.

When an area is no longer needed it may be released for use by other processes with the dbclose function. Dbclose parameters are a list of area names. All the areas in the list are closed. If the parameter list is omitted, all open areas are closed. After the dbclose is executed, all current records in closed areas cease to be current. If any area named in the parameter list is not open, error code 0101 is returned to error.status and error.count will contain the the number of errors detected.

When the process terminates (even abnormally), no dbclose is needed as all areas will be closed. If the dbclose function is executed on a temporary area, the data within the area is not lost and the area can be reopened and processed. When the process terminates, however, all temporary areas, open or closed, are lost.

d. Find.

The find function allows the user program to select a record from the data base and make it the current record of the run unit and, selectively, of the appropriate record and set types. The parameters are a record selection expression and a suppress code. The record selection expression is discussed in 1.b above. The suppress code is an octal code whose least significant bit indicates set supression and whose next least significant bit indicates record suppression. If set suppression is indicated, additional find parameters are permited, each of which is the name of set type.

Execution of a successful find function causes the selected record to become the current record of the process, the area in which it is located, the record type of the record and all set types in which it participates as an owner or member record. If record or set suppression is indicated, the object record does not become current for these types. When the list of set names is included, currency update is suppressed only in the named sets. After a find, the data fields of the record are not available, but its data base key can be derived (through the key function), a pointer to its area buffer is in "areaname" and a pointer to the appropriate record type structure is in "recname". The record can now be retrieved by the get function.

The following error codes may be returned into error.status by a find.

0301 The sought record is in an area which is not open.

0318 A record occurence along the search path of the find is in an area under the exclusive control of another process.

0302 A data base key was supplied or developed which is incompatable with the areas specified for a record of this type.

0307 An end of area or end of set condition was detected.

0326 No record in the area for selection through CALC key satisfies the record selection expression.

0322 Owner record selection is specified and the data base key given is for a record which does not participate in a set of the desired type.

0323 Relative selection was specified and the specified record cannot be in the desired area.

0310 A privacy breach was attempted.

0361 No call to the permit function has been made.

e. Get.

The get function is used to transfer the data values of the current record of the process into the process' buffers. Its parameters, which are optional, are item names from some record type. If the record type and item names are specified, only the items named are extracted. If the item names are not specified, all the items defined in

the subschema for that record are extracted. A get must be executed for a record before any of its item values can be examined.

The following error conditions may be returned to error.status for a get.

0513 The current record for the process is unknown.

0510 A privacy breach was attempted.

0520 A record name is specified and the current record of the process is not of that type.

0561 No call to the permit function has been made.

0554 Truncation of significance occured during conversion from the schema type to the subschema type for an item.

In all but the last case, no data is transfered to the user process.

f. Store.

The store function is used to create a new record occurence in the data base. It acquires space and a data base key for a new record occurence in the data base, causes the data items in the record's buffer to be used in initializing the record, inserts the record into all sets in which it is an automatic member and establishes a new set ocurrence of each set type for which the record is defined as an owner in the schema.

The parameters of the function are a record name, a suppress code and one or more set names. The record

name specifies the record type to be created. The suppress code and set names are exactly analogous to those of the find function. In order for the store to function properly, the subschema must include the following: the named record; the "data-base-identifiers" or set specified in the "LOCATION" mode clause of the record; at least one of the areas specified in the within clause for the record; all sets in which the record is defined as an automatic member; and all "data-base-identifiers", records and sets specified or referenced in the "SELECTION" and "KEY" clauses of the set member subentries in which the record is defined as automatic (see Ref. 2, section 3.4.0).

Prior to calling store, it is the user program's responsibility to insure the following is done. All data items in the record type buffer must be initialized. If multiple areas are defined in the "WITHIN" clause for the record with the "data-base-data-name-1" option (see Ref.2, Section 3.3.0) and the "LOCATION" mode is not direct, "areaid" must contain the desired area pointer. If the "LOCATION" mode is direct with the "data-base-data-name-1" option, keyname must have the appropriate data base key stored in it. If any automatic membership has a "SELECTION" method of "THRU CURRENT", the current record of the set type must specify the correct set. All data items mentioned in the selection clauses of the member entries which are automatic for the record and all data items mentioned in the "LOCATION" clause of the record entry must be initialized.

If an error occurs during the execution of a store, the new record is not created, no currency indicators are changed and an error code is returned to error.status. The following errors and codes can be encountered.

1201 The object record is to be stored in an area which is not open.

1221 A record occurence which is affected by the store function is in an area which is not open.

1209 The object record of the store or some record occurence affected by the store is in an area which is open for retrieval only.

1218 Some record occurence needed by the store for information (e.g. search paths) is in an area which is not available.

1212 No data base keys are available.

1211 No media space is available.

1202 A data base key passed by the user or generated via a "CALC" procedure is not valid.

1205 The record would violate a "DUPLICATES NOT ALLOWED" clause defined for one of the records or sets involved.

1225 For one of the set types involved a set occurence cannot be matched to the relevant set selection criteria.

1210 A privacy breach was attempted.

1227 A check clause applies and one of the data items did not pass.

1223 The area specified for the record is not one of

those in the record's "WITHIN" clause.

1224 The execution of the store statement would cause a set occurence to have records in both temporary and permanent areas.

1219 The value of an item cannot be converted to the type specified in the schema for that item.

g. Modify.

The modify function enables the updating of some or all of the data items defined in the sub-schema for a record and the changing of set occurences in which a record participates. The parameters, which are all optional, are a record name, a list of items in the record and set parameters identical to those of the insert function (see Section B.2.j). If the items are not specified, then every item in the record which is known to the sub-schema is updated, otherwise only the named items are updated. If the set names are specified, the action taken is equivalent to a remove function followed by an insert function for the named sets with the following exceptions. The record must be in an occurence of every set named prior to the modify function. The set membership in the named sets can be defined as mandatory or automatic, or both.

The object of the modify is the current record of the process. All data items to be updated and all items required for an insert on the named sets, must be initialized for the modify. If any of the modified data items are

sort control items for a set occurence in which membership is retained, the position within the set is modified accordingly. If any of the items changed are in a "SEARCH" key clause, the index is updated. The record becomes the current record of its record type and all sets it has membership in.

If an error occurs during a modify, no data base or currency changes are made and an error code is returned to error.status. The possible error conditions and the associated codes are as follows.

0803 One of the items changed is in a CALC key and the data base key would be altered, or an area number specified for owner record selection disagrees with the CALC key developed for the owner.

0825 A set occurence satisfying the specified criteria was not found.

0822 The record is not currently a member of every specified set.

0805 The insertion of the record into a set occurence would violate a "DUPLICATES NOT ALLOWED" clause.

0810 A privacy breach was attempted.

0827 A check clause was failed.

0821 Some record occurence affected by the modify is in an area which is not open.

0821 The object record or some record occurence affected by the modify is in an area which is not open for update.

0818 Some record occurence which is implicitly refer-

enced is in an area open for exclusive use by another process.

0819 A modified data item cannot be converted into the format used by the schema for the item.

0824 Insertion of the object record into some set occurence would cause that set occurence to have members in both temporary and permanent areas.

0861 No call to the permit function has been made.

In all cases, no change is made to the data base or to the currency indicators of the process.

h. Key.

The key function allows the extraction of the data base key for one of the current records. The function needs one parameter which may be a record, set or area name or the word "process". The function returns a pointer to a character array containing the data base key for the current record of the input parameter. The key should be treated as read only.

If an error occurs during the key function, a null pointer is returned and the error code is returned to error.status. The error conditions possible are no current record exists for the input parameter passed (code 1306) and no call to the permit function (code 1361).

i. Delete.

The delete function is used to destroy the current record of the process, releasing its data base key and storage, and to selectively delete all of the records which are members of set occurences owned by the current record of the run unit. The function requires a single integer parameter in the range zero to three with meaning as follows. A zero parameter causes deletion of the record if and only if it is not the owner of any non-empty set occurences. If the parameter value is one, the record is deleted, all optional members are removed from its set occurences and all mandatory members of its set occureneces are deleted. If the parameter is two, the action is identical to that of one except that, if any of the records whose membership is optional do not participate in set occurences owned by a different record, then they are deleted also. If the parameter value is three, then the record and every member of its set occurences are deleted. For any member record deleted, the deletion of the member records in that record's sets is decided as if that record were the object of a delete function with an identical parameter as that for the originally deleted record.

If an error occurs during the function, no records are removed or deleted and an error code is returned to error.status. The possible errors are as follows.

0230 A delete with parameter zero was attempted and the record owns a non-empty set occurence.

0213 The current record of the process is unknown.

0210 A privacy breach was attempted.

0221 One of the affected member records is in an area which is not open.

0209 The current record of the process or some affected record is in an area open for retrieval only.

0218 An implicitly referenced record is in an area which is open for the exclusive use of another process.

0208 The sub-schema does not know about all the record types which would be deleted or removed, or all of the set types of set occurences which would have records removed.

j. Insert.

This function causes the current record of the process to become a member of an occurence of the specified set types, providing it is defined as an optional automatic, optional manual or mandatory manual member of those sets. The parameters, which are optional, are a record type and one or more set names. Additional parameters may follow each set name depending on the selection criteria for the member entry of the object record's type. If the root set in the selection path has "DATA-BASE-KEY" specified with the "data-base-data-name-1" option (see Ref. 2, Section 3.4.0), a pointer to an array containing a data base key or an item name of type dbkey must be included. If the root set has the "CALC-KEY" option with the "data-base-data-names"

specified, item names or pointers to data whose type matches that of the corresponding items in the CALC key must be included. For each set after the root in the selection path which uses the "EQUAL TO data-base-data-name-4" option (see Ref. 2, Section 3.4.0), an item name or pointer to data which matches the type of the data item specified in the selection clause must be included. In addition to the explicit parameters above, all data items needed in the selection path as specified in the selection clause must be specified. If the owner record's "WITHIN" clause specifies multiple areas, "areaid" or the appropriate data item must be initialized to the appropriate area. See Reference 2, section 3.4.11 for a description of the selection clause. If a set name is specified with no additional parameters, then the set used is the current set of that type.

If the set names are specified, the record must not be in an occurence of any of the named set type. If no set names are specified, the record is inserted into the current occurence of each set type for which the record is defined as optional automatic, optional manual or mandatory manual provided the record does not already participate in a set of that type. After the insert, the record becomes the current record of every set to which it has been added.

If an error occurs during an insert, the data base remains unchanged, no currency indicators change and the appropriate error code is returned into error.status. The possible error conditions are as follows.

0713 The current record of the process is unknown.

0714 Set names are specified and the record is not defined as an optional automatic, optional manual or mandatory manual member of each of them.

0705 The record, when inserted, would violate a "DUPLICATES NOT ALLOWED" clause for some record or set involved.

0710 The current record of some set name specified in a "CURRENT" clause of a selection entry is unknown.

0716 The record is already in an occurence of a set explicitly specified or of every set implicitly specified.

0720 The record type was passed as a parameter and disagrees with the type of the current record of the process.

0721 A record occurence which is affected is in an area which is not open.

0709 The record inserted or some affected record is in an area which is open for retrieval only.

0718 A record occurence implicitly referenced by the insert is in an area which is not available.

0724 Insertion of the record into a set would cause the set to have members in both temporary and permanent areas.

0761 No call to the permit function has been made.

k. Remove.

This function is used to cancel the membership of the current record of the process in specified set occurences for which the record's membership is optional. The parameters, which are optional, are a record name and one or more set names. If the set names are specified, the object record must participate in an occurence of at least one of them and its membership in each of them is canceled. If no set names are specified, every optional membership in a set occurence for the record is cancelled.

If an error occurs during the remove, no set memberships are canceled, no currency information is affected and the error condition is returned into error.status. The following errors are possible.

1113 The current record of the process is not known.

1120 A record type parameter was passed and it disagrees with that of the current record of the process.

1115 The record is not defined as an optional member of any named set type.

1122 The record does not participate in at least one of the sets named; or if no sets are named, in at least one of the possible sets for which it is optional.

1110 A privacy breach was attempted.

1121 Some record affected by the remove is in an area which is not open.

1109 The current record of the process or some affected record is in an area which is not open for update.

1118 Some implicitly referenced record is in an area opened for exclusive use by another process.

1161 No call to the permit function has been made.

APPENDIX B. FILES ASSOCIATED WITH A SCHEMA.

A. Files in the Schema Directory.

Most of the files associated with a schema are contained in a directory bearing the name of the schema. This directory becomes the current directory for the schema DBM program. In the description of the files within the schema directory, the term "schemaname" indicates a variable portion of a file name which is replaced by the name of the particular schema when the files are named.

1. Source Description File.

The Source Description File contains the schema description in the source CODASYL DDL form. Its name is "s.schemaname".

2. Encoded Description File.

The Encoded Description File contains the compiled description of the schema. It contains data base names and encoded descriptions for the areas, records and sets in the schema. It is used principally by the schema DBM program in the initialization process. Its name is "des.schemaname".

3. Schema DBM Program.

The schema DBM program is the data base manager for the schema. It is comprised of the DBM skeleton routine compiled together with any data base procedures used in the schema. Its name is "dbm.schemaname".

4. Schema Library.

This file is optional and, when present, contains data base procedures unique to the schema. It is named "lib.schemaname".

5. Area Data Files.

These files contain the data for all the defined areas in the schema which are not designated as temporary. Their names are the same as the areas which they represent.

6. Area Data Base Key Files.

These files contain the byte offsets associated with each data base key for the areas which are not designated as temporary. The files are named by prefixing the area name by "k.".

7. Index Block File.

This file provides storage for all the indices used for set linkage in the data base. It is called "index.schemaname".

8. Open Lockout File.

This file is used by integrity routines to lock out other processes when setting up exclusive or protected access priveleges for a user process. It is created to initiate any open or close operation and removed when the operation is completed. The file is named "opendum".

9. Index Lockout File.

This file is used to lock out other processes while attempting to acquire an index from the index block file. It is handled in a manner analogous to the open lockout file. The file is name "indexdum".

10. Message Buffer File.

This file is used by the schema DBM program to assemble messages to the user program which are longer the 512 characters. Prior to storing a new message, the file is truncated to zero length.

B. Files in the Temporary Directory.

Certain files for a schema are stored in the UNIX temporary directory ("/tmp"). This directory has the characteristic that should a system crash occurs, all the files contained within it are lost. This directory is used to store files which are associated with the running of a process and therefore should be lost if the process is terminated by a system crash. The files are as follows.

1. Area Files.

These files are the area data files and area data base key files for all areas designated as temporary. The naming conventions for these files are identical to those for non-temporary data and data base key files with the exception that the process id (pid) of their user process is suffixed to the name.

2. Logical Usage Block File.

This file contains the logical usage block. This block is used during open and close operations to record the usage modes for the various areas currently in use. Its name is the same as the name of the schema.

APPENDIX C. DBM - THE DBM REQUEST PROCESSOR.

A. Introduction.

Dbm is a simple command language processor for schema level requests. It enables the data administrator to perform such functions as compiling a schema, moving data from schema to schema and garbage collection. It provides the user with a method of executing a program to utilize the schema. The functions of "ALTER", "DISPLAY" and "LOCKS" described in Ref. 2 are provided by different means. Namely, the UNIX "ed" and "list" functions are used, with privacy provided by the file access privacy of UNIX. The function "COPY" (for subschema use) is inapplicable since the C language DDL is not a proper subset of the schema DDL as was the case with COBOL. In addition, the cross checking of the sub-schema and the schema is done at execution time.

Prior to using dbm, the schema directory must exist (UNIX function "mkdir") and the schema source file should have already been created using "ed", the UNIX text editor. Dbm is called by entering "dbm pathname", where the pathname is a path ending with the schema name of the schema to be used. The program will respond in one of two ways: it will display "cannot access schema" or ">". The first response indicates that either the schema specified does not exist; access privacy prevents access; the schema is not a

directory; or a file called "s.schemaname" does not exist in the directory, where "schemaname" is the name of the schema. This response is followed by immediate termination of the program. The second response is the dbm prompt character and means that dbm is ready to accept commands.

B. Commands.

Upon recieving the prompt character, the user has the option of specifying any of six commands as follows.

1. Compile.

The compile command causes the schema to be compiled. The command format is "c" followed by a carriage return. The compilation process causes the scanning of the schema source file, "s.schemaname", and creation of the encoded schema description file, "des.schemaname", and the schema data base manager, "dbm.schemaname". If the necessary permissions are not present to create these files, dbm displays "cannot compile". If errors exist in the source file, they are displayed. In order to divert the error list, an optional path name parameter is allowed with the "c" command. If the specified file can be opened, the error listing is output to it, otherwise an error message is displayed at the user's terminal. When the "c" command is finished, the user receives a prompt. The compiler is currently a stub.

2. Move.

This command allows data to be moved from an old version of a data base to the current one. The command format is "m" followed by a path to a schema directory. Move will check the specified schema name to determine if it is a directory containing an encoded schema description and schema data base manager. If the schema is nonexistent or inaccessable, dbm will display "schemaname cannot be accessed", otherwise the data from the designated schema will be moved to the current schema. The data moved is selected by finding all area, record and set entries with common names and transfering the data which is associated with these common areas, records and sets. Area, record and set entries should have the same order in both schemas. All data presently in the current schema will be lost. If the move is unsuccessful, move produces error messages. After the move is completed, the user receives a prompt. The move function is currently a stub.

3. Execute.

This command causes the execution of a user program to access the data base. Its format is "x" followed by a path to a user program and the arguments for that user program. If the user program is inaccessable, nonexistant or not a program, dbm prints an error message and prompts. Otherwise dbm executes the user program and, upon its termination, prompts.

4. Garbage Collection.

This command allows waste compression in area data files for the data base. The format is "g" followed by a carriage return. This causes the following events for each area in the schema. The message "number of bytes wasted in areaname is NNN. Collect? (y or n)" to be displayed, where "areaname" is the area being processed and "NNN" is the area waste count. Entering "y" causes the area file to be recreated with all records written in ascending order of data base key and with all wasted space eliminated. Entering "n" causes the next area to be processed. When all areas have been processed, the user is prompted.

Due to the lack of a garbage collection facility in the schema dbm skeleton, frequent garbage collection may be necessary. Note garbage collection causes any assignment of data base keys designed to juxtapose related records to be reflected in the area data file as well.

5. Free.

If UNIX crashes during a dbm spawned function, certain files may be left in a state making restart impossible. The command "f" followed by carriage return causes this condition to be eliminated. The free command removes the files "opendum" and "indexdum" from the schema directory, if they exist, and scans the index block file, "index.schemaname", freeing any locked indices.

C. Interprocess Communication.

Whenver dbm must create a process, it uses the UNIX functions "fork" and "exec". The former causes a complete copy of the current process, called the child, to be created and the later causes the current process to be overlaid and replaced by the program specified. Dbm creates children as needed to do its work. Whenever a need may arise for the children to communicate with each other, dbm creates inter-process communications pipes. These pipes appear to be a pair of files, called the ends, each with an open file descriptor. One end of the pipe is open for reading and the other for writing. Since all children of a parent executing a pipe call have the pipe open also, the pipe can be used for passing data back and forth. Certain protocols must be observed, however. The pipe can only effectively be used for one way transmission since there is no protocol for preventing a process from reading its own tranmissions back before the intended receiver has a chance to read them. The receiver should close the writing end of the pipe, otherwise the receiver will wait forever if trying to read the pipe after the sending process has terminated. This phenomenon is caused by the fact that the process reading a pipe will go into wait state if any process, including the process doing the read, has its writing end of the pipe open. If no process has its writing end open (termination automatically closes all of a process' open files and pipes), a read on the pipe will return an end of file condition.

APPENDIX D. SCHEMA DESCRIPTION FILE FORMAT.

The schema description file contains the encoded schema description. The file is used by the dbm move command and in initializing the schema DBM program. Its format is described below.

A. Schema Entry.

The schema entry is headed by a null terminated string containing the schema name. Next is a privacy lock consisting of a null character, if no privacy lock is defined; or a one character type followed by a null terminatated string, if a privacy lock is defined. If the lock is defined, a lock type of "s" indicates that the string is a lock string and "p" indicates a lock data base procedure name.

B. Area Entries.

The area entries are preceded by a two byte number which is the number of areas areas and a two byte maximum record size. Each entry contains the following items.

The area name is a null terminated string. The temporary indicator is a one character flag which is equal to one for temporary areas and zero otherwise.

Fourteen data base procedure names are stored next. The first six names are procedures to be called when the open functions for retrieval, protected retrieval, exclusive retrieval, update, protected update, and exclusive update, are executed normally. The seventh name is a procedure to be executed when a close is executed normally. The final seven names are procedures corresponding to the first seven, but which are executed when errors occur. If a procedure is not specified for a function, a null string will appear in the file at the appropriate position.

Following the data base procedure names are the six privacy lock entries. These locks have the same format as the schema privacy lock. The six locks apply to the open function for retrieval, protected retrieval, exclusive retrieval, update, protected update and exclusive update respectively.

C. Record Entries.

The record entries include information generated by the member subentries of the schema's set entries as well as information from the schema's record entries. The record entries are preceded by a one byte number indicating the number of record types present. Each record entry contains the following data.

The record name is a null terminated string. It is followed by a two byte signed integer indicating record size

for records of this entry type. A one byte location mode is next. Additional location mode information may follow depending on the mode: for modes zero and seven, no additional information; for mode one, a one byte record index and a one byte item index; for modes two and three, two or more one byte item index numbers preceded by a one byte number indicating the number of indices present; for modes four and five, a null terminated string naming a data base procedure and two or more one byte item indices preceded by a one byte number indicating the number of indices present; and for mode six, a one byte set index. The location mode infromation is derived form the LOCATION clause of the record entry and the encoding matches that in "rlocmod" of a schema DBM record vector.

Following the location information is the area data derived from the record type's WITHIN clause. This consists of a one byte option code and area specifications. The area specification format depends on the option code: for code zero, a one byte area index; for code one, two or more one byte area indices preceded by a one byte number indicating the number of indices present; and for two, no further data. The encoding of the WITHIN information matches that of "rarea" in the schema DBM record vector.

Fourteen data base procedure names are stored next in the entries. The first seven are procedures to be called when the functions of "insert", "remove", "store", "delete", "modify", "find" and "get", are executed normally on records

of this entry's type. The second seven data base procedures are called when any of the above seven functions is executed and an error occurs. If no procedure is defined in the schema for a function, a null string replaces the function.

Following the data base procedure names are seven privacy lock entries. These locks have the same format as the schema privacy lock. The locks apply to the seven functions listed in the previous paragraph.

1. Member Data.

Each record entry has zero or more set membership entries following the record privacy locks. These entries are preceded by a one byte number indicating the number of membership entries present. The member entries for each record type appear in the same order as the set entries in the schema for which membership is defined. The contents of each membership entry is as follows.

The set name for the membership is stored as a null terminated string. Following the set name is a two byte series of flag bytes which correspond to the bits of "mflags" in a schema DBM member vector. The information contained in these bits is derived from the MEMBER clause, the KEY clause and the total number of SEARCH clauses defined in the schema.

Next is a one byte number indicating the number of items included in the primary key for the item. This number

is at most 16 and is zero if no key is defined. Following this value is the appropriate number of primary key element pairs. Each key pair consists of a one byte collating code and an item index. The collating code is zero if this element of the primary key is ascending and one if it is descending.

Following the primary key specification are up to seven search key strings (the exact number is recorded in the flag bytes above). Each search key string is a null terminated string of the item indices for the items in the search key.

Following the search key strings is the set selection data. If Format 2 of the SELECTION clause was used, this data consists of the name of a data base procedure. If Format 1 was used, the data is as follows. First is a one byte code indicating the root selection mode. The code corresponds to that in "mselflag" of a schema DBM member vector. The remaining root selection data depends on the root selection mode. For mode one and two, a one byte set index follows the mode. For mode three, there is no further root selection data. For mode four, the data is a null teminated string of two byte pairs each of which contains a record index and a set index.

The remaining set selection data for Format 1 consists of the number of "THEN THRU" clauses followed by the appropriate number of two byte selection pairs. Each pair

contains a set index and an item index. These pairs are the source data for "mssel" in a schema DBM member vector.

Following the set selection information are six data base procedure names. The first three procedures are called when the functions of "insert", "remove" and "find" are executed normally. The second three procedures are called when these same functions are executed and an error results. If no data base procedure is defined for a particular function, a null string will appear in its position.

Following the data base procedure names are three privacy lock entries in the same format as the schema lock. These privacy locks are for the functions described in the previous paragraph.

2. Item Data.

Each record entry has one or more item description entries following the set membership entries. The item entries are preceded by a one byte number indicating the number of items present. The item entries are stored in the same order that the items they represent appear in the record type being described. The contents of each item entry is as follows.

The first data in an item entry is the name of the item stored as a null terminated string. If the item is one that is not generated by an item sub-entry in the schema, the item name will be a null string. Following the name is

a one byte level number. A level number between one and 100 is generated by a schema item sub-entry; 101 is a forward link; 102 is a backward link; 103 is a link to owner; 104 is an owner's link to first member; 105 is an owner's link to last member; 106 is an owner's link to index and 107 is a CALC synonym link.

The remaining data depends on the level number. If the level number is between one and 100, inclusive, the next byte contains the data type; between 101 and 106, inclusive, the next byte is the set index; and for 107 no other data is needed. If a picture is defined for the item, it is stored next as a null terminated string.

Following the level and type data is a validity checking description. The validity checking description is a null terminated string which is encoded to fit the requirements of "icheck" in a schema DBM item vector. It is generated by the CHECK clause of a schema item sub-entry. If no validity check is defined for the item, the string is null.

Following the validity check description are three, two byte numbers representing the size (in bytes) of one occurence of the item; the number of occurences of the item in a record; and the starting byte number of the item within the record.

The names of six data base procedures are next. The first three are names of procedures to be called when the

functions of "store", "get", and "modify" are executed normally. The second three procedures are called when these functions are executed and an error occurs. If a procedure is not defined for a function, a null string will appear in its place.

Following the data base procedure names are three privacy lock entries in the same format as the schema lock. These locks apply to the functions mentioned in the previous paragraph.

D. Set Entries.

Following the record entries are the set entries. These entries are generated by set sub-entries in the schema. The entries are preceded by a one byte number indicating the number of sets defined. The contents of each entry is as follows.

The first element in each entry is the set name stored as a null terminated string. The set name is followed by a one byte code which corresponds to the lower order byte of "sflags" in a schema DBM vector and describes OWNER, SET IS and ORDER clauses of the set sub-entry.

Next is a pair of bytes indicating the owner record. The first byte is the owner record's index and the second is the item index of the first item in the owner record having to do with the set. Following the owner record data is a one byte number indicating the number of member records and

three byte member descriptions present. Each member description consists of the record index of the member; the index of the set membership vector in the schema DBM record vector for the record; and the item index of the first item in the record having to do with this set. The order of the member descriptions is alphabetical by member record name.

Following the member descriptions are four data base procedure names stored as null terminated strings. The first two are names of procedures to be called when the "insert" or "remove" functions are executed normally. The last two represent the same functions, but are called when an error occurs. If a procedure is not defined in the schema for a function, a null string appears in that place.

Following the data base procedure names are three privacy locks. These locks are of the usual format. They lock the functions of "insert", "remove" and "find", respectively.

APPENDIX E. INTERPROCESS MESSAGE FORMATS.

A. Messages Received by the Schema DBM.

Messages received are read into "smesin", a character buffer of length 512. The first byte of the message is a function code. The remainder of the message will vary depending on the function code. The message is terminated by a mark, which is ten bytes of the octal code 0252. In the message descriptions that follow, the function code is included as part of the description heading.

1. Initial Call Message (Code 0).

The initial call is made by the user to request validation of his sub-schema and to establish his access permissions. Immediately following the function code is a null terminated string containing the schema name. After the schema name is a null terminated string containing the privacy key for the schema. Following the schema entries are the area entries.

a. Area Entries.

The area entries are preceded by a one byte number indicating the number of areas in the sub-schema. Each entry consists of seven null terminated strings. The first string is the area name, the other six strings are

privacy keys and may be null strings. The privacy keys specified are for retrieval, protected retrieval, exclusive retrieval, update, protected update and exclusive update respectively.

b. Record Entries.

The record entries are preceded by a one byte number indicating the number of records in the sub-schema. Each entry consists of seven null terminated strings, an encoded member list and an encoded item list. The first string is the record name and the remaining six are privacy keys and may be null strings. The privacy keys are for insert, remove, store, delete, modify and find respectively.

An encoded member list is headed by a one byte number indicating how many member entries follow. Each member entry consists of four null terminated strings. The first string is the name of the set and the remaining three are privacy keys and may be null. The privacy keys are for insert, remove and find respectively.

An encoded item list is headed by a one byte number indicating how many item entries are in the list. Each entry has a one byte entry code followed by four null terminated strings. The first string is the item name and the rest are privacy keys and may be null. The privacy keys are for store, get and modify. The remainder of the item entry varies depending on the entry code specified below.

(1) Atomic Item (Code 0). No further fields exist in the item entry for an atomic item.

(2) Vector (Code 1). A one byte number indicating the number of occurences of the item follows the privacy keys.

(3) Repeating Group (Code 2). A pair of one byte numbers follows the privacy keys. The first number indicates the number of subsequent item entries in the repeating group and the second indicates the number of occurences in the group.

c. Set Entries.

The set entries follow the record entries. They are preceded by a one byte number indicating the number of set entries. Each set entry consists of four null terminated strings. The first string is the set name and the remaining three are privacy keys and may be null. The privacy keys are specified for insert, remove and find.

2. Open Message (Code 9).

The function code is followed by a one byte mode, which uses the same encoding as the C dbopen function (see Appendix A, Section B.2.b). The remainder of the message consists of one byte area index numbers. No area numbers should be included if every area known to the subschema is to be opened.

3. Close Message (Code 1).

The function code is followed by a list of one byte area index numbers. No area numbers are included in the message if all open areas are to be closed.

4. Find Message (Code 3).

The function code is followed by a one byte selection type and selection codes. The possible selection types and their corresponding record selection codes are:

a. Explicit Key.

Code zero indicates direct access and the selection code will be a four byte data base key.

b. Owner Record.

Code one indicates selection of the owner record for the set of the specified type that the specified record belongs to. The first selection code is a one byte set index and the second is a four byte data base key.

c. Relative Area.

Type code two specifies relative selection in the designated area. The first selection code is a one byte criterion with zero, one, two and three meaning next, previous, first and last, respectively; and four, five, six and seven meaning next, previous, first and last of a specified record type. If the criterion is four through seven, the

second selection code is a one byte record index. The last selection code is a four byte data base key.

d. Relative Set.

Type code three specifies relative selection in the designated set. The first selection code is a one byte set index. The remaining selection codes are identical to those for type code two.

e. CALC Key.

Type code four indicates hash key selection. The first selection code is a one byte record index. The remaining codes are item triples for all the items of the specified record type which are know to the sub-schema and are not associated with an "OCCURS" clause. An item triple consists of an item specification, a one byte data type code, and a data value of the specified type. An item specification code consists of a one byte item index followed by zero or more one byte subscript values as appropriate. The data type codes are one for integer, two for single precision floating point, three for double precision floating point, four for null terminated string and five for data base key.

f. Duplicate CALC Key.

Type code five indicates selection of the next record with a hash key duplicating the hash key of the specified record. The selection code is a four byte data

base key.

g. Current Set Data Value.

Type code six indicates that the first record of the specified type which matches the specified item values in the specified set occurence is to be selected. The first selection code is the one byte record type index of the record type to be selected. The second selection code is a one byte set type index. The third selection code is a four byte data base key of a record which belongs to the set occurence to be scanned. The remaining selection codes are zero or more item triples.

h. Selected Set Data Value.

Type code seven indicates that the first record of the specified type which matches the specified item values in the set occurence selected through the specifed record type's member subentry "SELECTION" clause is selected. The first two selection codes are identical to those in the preceding paragraph.

The third selection code is a one byte number which indicates the number of path selection codes which follow. The path selection codes are item quadruples. An item quadruple consists of a one byte record index, an item specification, a one byte data type code, and a data value of the type specified in the data type code. The remaining selection codes are zero or more item triples for items in

the specified record type.

i. Current Set Duplicate Value.

Type code eight indicates that the record to be selected is the next record (if any) which is of the same type as the specified record; in the set of the specified type; and matches the specified record in the specified items. The first selection code is a four byte data base key. The second selection code is a set type index. The remaining selection codes are item specifications.

5. Get Message (Code 5).

Following the function code is a four byte data base key. The remainder of the message consists of item doubles for the items the user program desires. An item double consists of an item specification and a one byte data type. An omitted subscript in a data specification means every occurence of the vector or repeating group is desired. A double for a repeating group has a data type of zero. The doubles for the elements in the repeating group must be immediately following the repeating group's double.

6. Store Message (Code 12).

The remainder of the message is a one byte record type index.

7. Response to Request for Data Message (Code 100).

This message provides data requested in a Request for Data Message sent by the schema DBM. The function code is followed by an item specification for each requested item consisting of a one byte record type index, a one byte item index and a one byte data type. The requested data follows the item type specifications in exactly the same order as in the Request for Data. For group items, the order of appearance of the subordinate items of the group in the item specifications is the order the items must appear within each occurence of the group item in the data portion of the message. The other data may be area or set type indices or data base keys.

8. Insert Message (Code 7).

The function code is followed by a four byte data base key of the record to be inserted. The remainder of the message consists of set specifier pairs. A set specifier pair consists of a one byte set type index followed by a four byte data base key indicating the current record in the current occurence of the set of the type specified.

9. Remove Message (Code 11).

Following the function code is a four byte data base key. The remainder of the message is zero or more one byte set type indices.

10. Modify Message (Code 8).

Following the function code is a one byte number indicating how many set memberships are being modified, followed by the required number of set specifier pairs. The remainder of the message consists of zero or more item triples.

11. Delete Message (Code 2).

Following the function code is a four byte data base key. The remainder of the message is a one byte deletion code with the same values as for the parameter of the C "delete" function (see Appendix A, Section B.2.i).

B. Messages Transmitted by the Schema DBM.

Messages transmitted are in response to messages received and fall into two categories: normal responses and error messages. The first byte of the message is a response code and is zero for normal responses and equal to the error code for error messages. The format of the responses, after the first byte, varies depending on the previously received message (for normal responses) or on the error type (for error messages). These formats are detailed below.

1. Normal Responses.

a. Initial Call Message.

The normal response to an initial call is an encoded schema description for the user. This description is a series of one byte numbers each of which represents the index number associated with the data base names (except the schema name) in the Initial Call Message. For areas, this is the index number of the area; for records, the index number of the record followed by the index number of each data item or data aggregate; for sets, the set number.

b. Find Message.

The normal response to a Find Message contains the information necessary to establish currency for the selected record. This information consists of a four byte data base key; a one byte record type index indicating the type of the record; and zero or more one byte set type indices indicating the set types for the set occurences, in which the record participates.

c. Store Message.

The normal response to a Store Message is a Request for Data Message. This message requests the data needed to perform the store function. The message is composed of request entries each prefixed by a one byte request type code. The request entry formats are listed below along with their request type codes.

(1) Data Item Request (Code 0). This request entry consists of an item specification. A repeating group index is an implied request for all subordinate elements in the repeating group known to the sub-schema. If a subscript is missing, the data in all occurences of the relevant element is requested.

(2) Area Index Request (Code 1). This request consists of the request code alone. It requests the contents of "areaid" in the user program.

(3) Data Base Key (Code 2). This request consists of the request code alone. It requests the data base key associated with "keyname" in the user program.

(4) Current of Set. This request consists of a set type index. It requests the data base key of the current record of the specified set type.

d. Request for Data Message.

The normal response to a Request for Data is identical to that for a find. The record information passed is for the record just stored.

e. Messages with a Null Response.

The normal response to certain messages is a response code only. These messages are Open, Close, Insert,

Modify, Remove and Delete.

2. Error Messages.

a. Invalid Sub-schema (Code 60).

This response is the result of a mismatch between the schema and the sub-schema presented in the user program's Initial Call Message. After the error code is a one byte first error type: zero for schema entry; one for area entry; two for record entry; and three for set entry. Following this code are three, one byte entries giving the number of the first erroneous entry in area, record and set entries respectively.

b. Area Already Open (Code 28).

This response to an Open Message has a one byte error count following the response code.

c. Truncation of Data (Code 54).

This response to a Get Message has identical format to the normal response to a Get Message except for the response code.

d. Messages with Error Code Only. The remaining error responses consist of an error code only as follows.

(1) Data Base Key Invalid (Code 2).

(2) Data Items Invalid or Inconsistent (Code 4).

(3) Violation of DUPLICATES NOT ALLOWED clause (Code 5).

(4) End of Set or Area (Code 7).

(5) Invalid Record or Set Index (Code 8).

(6) Attempted Update on Retrieval Only Area (Code 9).

(7) Privacy Breach Attempted (Code 10).

(8) Media Space not Available (Code 11).

(9) Data Base Key not Available (Code 12).

(10) Insert into Mandatory Automatic Set (Code 14).

(11) Remove out of Mandatory Set (Code 15).

(12) Insert into Set with Existing Membership (Code 16).

(13) Implicitly Referenced Area not Available (Code 18).

(14) Affected Area not Open (Code 21).

(15) Illegal Area Index (Code 23).

(16) Set Occurence would Span Temporary and Permanent Areas (Code 24).

(17) No Set Occurence Satisfies Specified Arguments (Code 25).

(18) No Record Satisfies Record Selection Expression (Code 26).

(19) CHECK Clause Violated (Code 27).

(20) Usage Mode Conflict with Other Processes (Code 29).

(21) Unqualified DELETE on Owner of a Non-empty Set (Code 30).

(22) No Initial Call Message (Code 61).

(23) Indecipherable or Unproccessable Message (Code 100).

APPENDIX F. DBM SKELETON PROGRAM.

The skeleton program is identical for every schema DBM. Compiled schema DBM's differ only in the values associated with certain DEFINE'd constants controlling array sizes, in the initialization of certain arrays, and in the data base procedures which are included in the compiled version. When the schema DBM is executed, it initializes its tables from the schema description in the Schema Description File. These tables drive the processing of the data base. The schema DBM concurrently reads the user's sub-schema description from the interprocess communication pipe. The sub-schema description is validated and index numbers are produced to allow translation of user requests and data into system requests and data.

This Appendix describes the data organization of the skeleton. Documentation for each service routine and utility routine is contained in the source program listings. Listings and machine readable source of the DBM skeleton can be obtained by contacting the Department of Computer Science (Code 52Rs). For an explanation of the values associated with DEFINE'd constants mentioned in this Appendix see Appendix H. For a description of a user's view of the service routines, see Appendix A.

A. General Tables.

Certain tables and buffers are available for use by other tables.

1. The character buffer, "scharbuf", is dimensioned by a DEFINE'd constant. It is used to stored character strings and other relatively short variable length data.

2. "Procname" is an array of strings containing the names of the data base procedures. "Procpoint" is an array of function pointers pointing to the functions defined in "procname". These arrays are used to set up the data base procedure pointers used in other tables. Both "procname" and "procpoint" are dimensioned by a DEFINE'd constant.

3. The privacy vector array ("pvec"), dimensioned by a DEFINE'd constant, is used for data item and data aggregate privacy information. Its elements are structures of type "privect". The format of a "privect" structure is

```
struct privect {
    char ptype;      // type of privacy lock
    char *plock;     // pointer to privacy lock
    }
```

The "ptype" code is "s" if the "plock" pointer points to a string, and "p", if a data base procedure is indicated.

4. The record buffer array, "srecbuf", contains all the record buffers for areas, records and sets.

B. Organization for Area Management.

The schema DBM contains an array of structues of type "areavect", dimensioned by a DEFINE'd constant, called "avec". Each structure in the array is used to describe an area in the data base. The format of an "areavect" structure is as follows:

```
struct areavect {
    int aflags;       // see below
    int ause;         // usage count of last reference
    char *adatapath;// pointer to path to data file
    int adatades;     // file descriptor for data file
    char *akeypath; // pointer to path to key file
    int akeydes;      // file descriptor for key file
    char acrecloc[3];    // location of current record
    char *acurrec;    // pointer to current rec buffer
    int acurkey[2]; // first key # in current key buff
    char akeybuf[768];// buffer for db key mappings
    int (*provec)()[14];// pointers to db procedures
    int apflags;      // permission flags for functions
    int awaste;       // current waste count
    }
```

"Aflags" is formed by a bit-wise OR of the following octal codes:

TEMP	0100000	The area is temporary
PHSOP	040000	The files are physically open
KEYBMOD	020000	Current key block modified
CRECMOD	010000	Current record modified
KEYBVAL	04000	Key block is valid
CRECVAL	02000	Record buffer is valid
CRECSIZ	01000	Current record has increased size
KNOWN	0400	Area is known to the sub-schema
RETRV	01	Area open for retrieval
PRETRV	02	Area open for protected retrieval
ERETRV	03	Area open for exclusive retrieval

UPDT	04	Area open for update
PUPDT	05	Area open for protected update
EUPDT	06	Area open for exclusive update

The "apflags" are set when the user sub-schema is validated. These flags indicate the functions allowed the user for the area as follows:

0100000	Retrieval
040000	Protected retrieval
020000	Exclusive retrieval
010000	Update
04000	Protected update
02000	Exclusive update

C. Logical Usage Block.

The logical usage block records the current usage mode for each area in the data base currently being used by any schema DBM. The logical usage block is organized into two byte integer entries, one for each area in the data base. Each two byte entry is divided into four fields: bit 15 is the exclusive use bit; bits 14 through ten form a count of retrievers; bits nine through five, a count of protected retrievers; and bits four through zero form a count of updaters. The logical usage block can record up to 31 users in each category. If a schema DBM has an area open for a protected mode, the updater count is set to 31.

D. Organization for Record Management.

The schema DBM contains an array of structures of type "recvector". Each element of the array is the record vector for a record type defined in the schema. The format of a "recvector" structure is:

```
struct recvector {
    int rflags;      // see below
    char rlocmod[3]; // see below
    char rarea[3];   // see below
    int (*rprovec)()[14]; // pointers to db procedures
    char rnumsets;   // number of set types for record
    struct member *rsets;    // pointer to member vectors
    char rnumitem;   // number of items in record
    struct itemvect *ritems; // pointer to item vectors
    int rpflags;     // permission flags for functions
    char *rcurrec;   // pointer to current record buffer
    }
```

"Rflags" is currently used only to indicate whether or not the sub-schema knows about the record type. The octal code KNOWN (0400) is used for this function.

"Rlocmod" is derived from the LOCATION clause of the schema RECORD entry (see Section 3.3.4. of Ref. 2) and is interpreted as follows. Character zero gives the location mode: zero for DIRECT with key passed as a parameter; one for DIRECT with key stored in a record; two for CALC using the standard key transformation with no duplicates; three for CALC using the standard key transformation with duplicates allowed; four for CALC using a data base procedure with no duplicates; five for CALC using a data base procedure with duplicates allowed; six for VIA a set; and seven for SYSTEM mode. The last two bytes in the "rlocmod" vary in meaning depending on the mode. For mode zero, bytes one

and two are unused. For mode one, byte one contains the record type and byte two, the item index of the data item which holds the data base key. For modes two through five, bytes one and two hold a pointer to the randomizing key description. A randomizing description is a null terminated series of bytes, the first containing the item index of the key link item and subsequent bytes containing the item indices of the fields of the randomizing key. For modes four and five, the randomizing key description is headed by a pointer to the appropriate data base procedure. For mode six, byte one contains the set index of the set to be consulted and byte two is unused.

"Rarea" is derived from the WITHIN clause of the schema RECORD entry and is formatted as follows. Byte zero is the WITHIN option code and has the following interpretation: zero, all records are within a single area; one, multiple areas are possible (selected by a user input value); two, the area will be the area of the owner of the set of a specified type in which the record participates. The values of bytes one and two of "rarea" vary depending on the WITHIN option: for zero and two, byte one contains an area index number and byte two is unused; for one, bytes one and two contain a pointer to a WITHIN criteria. A WITHIN criteria is a null terminated string of bytes each containing one of the allowed area numbers for this record.

1. Member Vectors.

Each record vector contains a pointer ("rsets") to an array of set membership vectors if it participates in any sets. A set membership vector is a structure of type "member". The format of the "member" structure is:

```
struct member {
    char msetnum;      // set index for this entry
    int mflags;        // see below
    int morder;        /* flag bits for key item
                          0 = ascending, 1 = descending*/
    char *mpkey:       // pointer to items for prime key
    char **mskey;      // pointer to SEARCH index pointers
    char mselflag;     // root selection flag
    int *mselid;       // pointer to root selection id
    int (*mssel)();    // pointer to set selection spec
    int (*mprovec)()[6];// pointer to db procedures
    char mplfags;      // permission flags for functions
    }
```

The "mflags" for a member entry is formed by a bit-wise OR of the following codes:

MMAND	0100000	Membership is mandatory
MAUTO	040000	Membership is automatic
MLINK	020000	Member is linked to owner
MSSEL	010000	Set selection by db procedure
MPKEY	04000	Primary key is defined
MPRKEY	02000	RANGE option
MPFKEY	01000	Duplicates first
MPLKEY	0400	Duplicates last
MPDKEY	0200	Duplicates arbitrary
MPNKEY	0100	Nulls allowed
MKNOWN	040	Membership is known to sub-schema

Whenever any of MPRKEY through MPNKEY are set, MPKEY must be

set. Additionally, the low order three bits of "mflags" contain the number of secondary indices defined to support SEARCH keys for this membership.

The pointer "mskey" points to a string of pointers dimensioned by the count stored in "mflags". Each pointer points to a null terminated string. The first byte of this string indicates whether duplicates are allowed; the second byte is the item index for the owner record item linking the search index; and the remaining bytes are item indices, each representing a field in the SEARCH key for the search index. Duplicates are allowed if the first byte of the string is a one and not allowed if it is a two.

"Mpflags" is set when the sub-schema is validated. These flags indicate the functions allowed the user for the record/set pair as follows:

0200	Insert
0100	Remove
040	Find.

When "mflags" has MSSEL set, "mssel" is a pointer to a data base procedure for set selection and "mselflag" and "mselid" are unused. If MSSEL is not set, "mselflag" is a code describing the root set selection in the set selection chain for this member. The possible values of "mselflag" are as follows: one for singular sets; 2 for current of set type; three for through data base key; and four for through CALC key. The data in "mselid" depends on the value of

"mselflag". For singular sets and current of set selection, "mselid" contains the set index of the root set. For selection by data base key, "mselid" is not used. For selection by CALC key, "mselid" is a pointer to a null terminated string of character pairs which are the record and item indices of the items to be used in forming the CALC key.

When set selection is not by a data base procedure and the number of THRU clauses in the SELECTION clause for this member entry is greater than one, then a selection chain exists and "mssel" is a pointer to a null terminated string of byte pairs. Each pair in this string describes the set selection for one of the successive set types in the set selection chain. Each pair consists of the index of the next set in the chain and the index of the data item which must be matched in the owner record.

2. Item Vectors.

Each record contains a pointer ("ritems") to an array of item description vectors. Each element of the array is a structure of type "itemvect" and describes one of the fields appearing in the record. A field may be in a record for CALC key linkage, for set linkage or as a result of a data sub-entry in the record's source description. The format of an "itemvect" structure is:

```
struct itemvect {
    char *iname;      // pointer to name of item
    char ilevel;      // item level
    char itype;       // type of data represented
    char *ides;       // data description pointer
    char *icheck;     // pointer to validity check
    int isize;        // size of one occurence of item
    int inocc;        // number of occurences of item
    int isbyte;       // starting byte within record
    int (*iprovec)()[6];// pointers to db procedures
    struct privector *ipvec;// pointer to privacy locks
    char ipflags;     // item privacy flags
    }
```

The "ilevel" entry specifies the level number of the item. A level number beween one and 100 indicates the item was generated by an item sub-entry; level 101, a forward chain link for a set; 102, a backward link; 103, a link to owner; 104, an owner's link to first member; 105, an owner's link to last member; 106, an owner's link to index; and 107, a CALC synonym link.

"Itype" is the data type code for the item: zero for repeating groups; one for a PICTURE'd character string; two for a PICTURE'd numeric string; three for a binary integer; five for a single precision floating point number; six for a double precision floating point number; seven for a character string; eight for a bit string; and nine for a data base key.

If the item is a set link, "ides" is the set index for the set. If the item has a picture specified, "ides" is a pointer to a character string containing the picture (see Ref. 2, Section 3.3.8 for a description of PICTURE'd data). In other cases, "ides" is unused.

"Icheck" is a pointer to a validity checking description for the item. The first character in the description is a flag byte. If the high order bit of the flag byte is on, the picture is used as a check. If bit six is on, a data base procedure is used as a check. If bit five is on, check values are used. If a data base procedure is specified, a pointer to the procedure is stored immediately after the flag character. If check values are specified they are stored at the end of the validity checking description. Check values consist of a series of check entries seperated by ASCII comma characters and terminated by a null byte. Each check entry is either a literal of the same format as the item or a pair of such literals separated by an ASCII dash character.

The "ipflags" are set when the sub-schema is validated. The octal codes and function permissions are:

0200	Store is permitted
0100	Get is permitted
040	Modify is permitted

E. Organization for Set Management.

The schema DBM program contains an array of set vectors. Each set vector describes one of the set types defined in the schema and is a structure of type "setvect". The format of a "setvect" structure is:

```
struct setvect {
    int sflags;         // see below
    char sowner;        // owner record type
    char sfitem;        // index of 1st item for set
    char *smemb;        // pointer member description
    int (*sprovec)()[4]; // pointers to db functions
    char spflags;       // function permission flags
    char scurown[3]; // db key of current owner rec
    char *scurrec;      // pointer to current record buf
    }
```

The value of "sflags" is formed by a bit-wise OR of the following octal codes:

KNOWN	0400	Set type is known to sub-schema
SYSTEM	0200	Singular set
DYNAMIC	0100	Dynamic set type
PRIOR	040	Members contain backward links
INDEXED	020	Primary set order is via an index

The lower four bits of "sflags" indicate the order criteria for the set: zero, the order is immaterial; one, new records are inserted on the front of the set; two, new records are inserted at the end of the set; three, new records are inserted after the current record of the set; four, new records are inserted prior to the current record; five through 11, a sorting order. Five indicates sorted by data base key; six, sorted by record names and then by member keys; seven, sorted by the member record keys with relationship between records of different types immaterial; and eight through 11 indicate sorted by member keys (this implies that the format of each member record's keys is the same). The last four codes specify duplicate processing: eight, duplicates are allowed; nine, duplicates are first; ten, duplicates are last; and 11, duplicates are not

allowed. Items in the owner record dealing with the set are assumed to be stored contigously.

"Smemb" points to a null terminated string of bytes indicating the member record types for the set. The string is made up of three byte entries. The first byte is the member record index; the second is the membership vector index of the member record for this set; and the third is the item vector index of the first item in the record dealing with this set. All items having to do with the set are assumed to be stored contiguously in the member records.

The "spflags" are set when the sub-schema is validated. These flags indicate the function allowed the user for the set:

0100	Insert is allowed
040	Remove is allowed
020	Find is allowed

APPENDIX G. DIFFERENCES IN THE SCHEMA DDL.

This Appendix gives a detailed description of the differences between the DDL in the UNIX DBMS and that described in Ref. 2. The Appendix is organized in parallel with Section 3 of Ref. 2 and section references below are sections in Ref. 2 unless otherwise noted. The meta-language used to describe entries is identical to that of Ref. 2 with the exceptions that no distinction is made between required or optional words and that options enclosed in brackets are separated by virgules ("/") in lieu of being on separate lines.

A. Words.

The rule in section 3.0.3 for forming words applies to the DDL. However, when validating a sub-schema, the DBMS considers upper and lower case letters to be equivalent and considers underscore ("←") a synonym for hyphen ("-").

B. Schema Entry (Section 3.1.0).

The "ON [ERROR DURING]" clause is not supported. In the "PRIVACY LOCK" clause, only the "[FOR COPY]" option is supported. Specifying alternate privacy locks for the same function is not supoported.

C. Area Entry (Section 3.2.0).

Specifying alternate privacy locks for the same function is not supported.

D. Record Sub-entry (Section 3.3.0).

In the "PRIVACY LOCK" clause, specifying alternate privacy locks is not supported. Specifying a data base procedure for area selection in the "LOCATION MODE" clause is not supported.

E. Data Sub-entry (Section 3.3.0).

In the "TYPE" clause, the only arithmetic types supported are "BINARY FIXED", "BINARY FLOAT 4" and "BINARY FLOAT 8". The word "BINARY" is assumed if missing, and "4" is assumed if neither "4" nor "8" is specified. If the "TYPE" clause uses the "BIT integer-3" option, "integer-3" must be a multiple of eight. Since all records must be fixed length, the "OCCURS data-base-identifier-1 TIMES" option is not supported. "RESULT" and "SOURCE" items, both virtual and actual, are not supported. The "FOR {ENCODING/DECODING}" clause is not supported. Specifying alternate privacy locks on the same function is not supported.

F. Set Sub-entry (Section 3.4.0).

The "TEMPORARY" option of the "ORDER" clause is not supported. The "[INDEXED [NAME IS index-name-1]]" clause is not supported. Specifying alternate privacy locks for the same function is not supported.

G. Member Sub-entry (Section 3.4.0).

In the "RANGE KEY" clause, no more than sixteen data-base-identifiers may be specified. The "DUPLICATES NOT ALLOWED FOR" clause is not supported. No more than seven "SEARCH KEY" clauses can be specified. In the "SEARCH KEY" clause, the "USING" phrase is not meaningful since all search keys are implemented using indices. In Format 1 of the "SET SELECTION" clause, the "DATA-BASE-KEY EQUAL TO data-base-identifier-1" and "CALC-KEY EQUAL TO data-base-data-name-2 [data-base-data-name-3] ..." options are not supported. In the same clause, the only form of the "THEN THRU" phrase supported is without the "EQUAL TO" option. Specification of alternate privacy keys on the same function is not supported.

APPENDIX H. CONSTANT FILE CONTENTS.

As mentioned in Section III.L.1, a constant file must be created by the DBMS compiler to allow the skeleton program to be transformed into the schema DBM for a particular data base. This file dimensions the tables and arrays of the schema DBM and initializes arrays for the processing of data base procedures. The specific tables and arrays are described below.

A. The Character Buffer.

The character buffer, "scharbuf", is utilized for storage of character strings and several other types of variable length data. This character buffer must be large enough to contain the schema name; the path names to all schema files (including temporary ones); the item names of every item in every record; the primary key, search key and selection data for every membership vector; the data and validity check descriptions of every item vector in every record; and the member record string for every set vector.

B. Data Base Procedure Table.

This table is composed of two arrays. The first array, "procname", is an array of character strings and must be initialized with the name of every data base procedure mentioned in the schema. The second array, "procpoint", is an array of function pointers which must be initialized to point to the data base procedures listed in "procname". The references to data base procedures in "procpoint" cause the C compiler to load these functions into the schema DBM.

C. Privacy Vector Array.

This array of structures of type "privector" must be dimensioned large enough to hold the maximum number of item privacy locks defined in any one record entry. The format of a "privector" structure is described in Appendix F, Section A.3.

D. Record Buffer Array.

This is a character array which is used to provide record buffers for the various areas, records and sets. Its dimension must be the number of areas and sets times the maximum record size plus the size of each individual record.

E. Area Vector Array.

This array of structures of type "areavect" must be dimensioned large enough to provide one area vector for each defined area. An "areavect" structure is described in Appendix F, Section B.

F. Record Vector Array.

This array of structures of type "recvector" must be dimensioned large enough to provide one record vector for each defined record type. A description of the "recvector" structure is contained in Appendix F, Section D.

G. Member Vector Array.

This array of structures of type "member" must be dimensioned large enough to provide one member vector for every record membership defined in every set. A description of the "member" structure is contained in Appendix F, Section D.1.

H. Item Vector Array.

This array of structures of type "itemvect" must be dimensioned large enough to provide an item vector for every item in every record type. A description of the "itemvect" structure is contained in Appendix F, Section D.2.

I. Set Vector Array.

This array of structures of type "setvect" must be dimensioned large enough to provide a set vector for every set defined in the schema. A description of the "setvect" structure is contain in Appendix F, Section E.

LIST OF REFERENCES

1. Ritchie, D. M., "The UNIX Time Sharing System", Communication of the ACM, Vol. 17, No. 7, p. 365-375, July, 1974.

2. Conference on Data Systems Languages, CODASYL Data Description Language JOD, U. S. Department of Commerce, June, 1973.

3. Conference on Data Systems Languages, CODASYL Data Base Task Group April 71 Report, ACM, 1971.

4. McDonald, N., Stonebraker, M. and Wong E., "Preliminary Design of INGRESS: Part I", Electronics Research Lab., Univ. of California, Berkeley, ERL-M435, April 10, 1974.

5. Held, G. and Stonebraker M., "Storage Structures and Access Methods in the Relational Data Base Management System, INGRESS", Proc. ACM Pacific Conf., San Francisco, April 17-18, 1975.

6. Earnest, C. P., "A Comparison of the Network and Relational Models", Computer Sciences Corporation, El Segundo, Calif., April, 1974.

7. Date, C. J., An Introduction to Data Base Systems, Addison-Wesley, 1975.

8. Huits, M., "Requirements for Languages in Data Base Systems", p. 85-109 in Data Base Description, Douque, B. C. M. and Nijssen, G. M. (eds.), North Holland/American Elsevier, 1975.

9. Codd, E. F. and Date, C. J., "Interactive Support for Non-Programmers: The Relational and Network Approaches", Proc. 1974 ACM-SIGMOD Debate, "Data Models: Data Structure Set versus Relational", Rustin, R. (ed.), Ann Arbor, Mich., May 1-3, 1974.

10. Ritchie, D. M., C Reference Manual, Bell Telephone Laboratories, 1975.

11. Kaimann, R. A., Structured Information Files, Wiley and Sons, 1973.

12. Bachman, C. W., "The Programmer as Navigator", Comm. ACM, Vol. 16, No. 11, p. 653-658, Nov. 1973.

13. Martin, J. T., Computer Data Base Organization, Prentice-Hall, 1975.

14. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks", Comm. ACM, Vol. 13, No. 6, p. 377-387, June, 1970.

15. Codd, E. F., "Relational Completeness of Data Base Sub-languages", Courant Computer Science Symposia 6, "Data Base Systems", New York, May 24-25, 1971, Prentice-Hall, 1971.

16. Date, C. J., "Relational Data Base Systems: A Tutorial", Proc. Fourth International Symposium on Computer and Informational Sciences, Miami Beach, Florida, Dec. 14-16, 1972, Plenum Press, 1972.

17. Conference on Data Systems Languages, CODASYL Data Base Task Group October 69 Report, ACM, 1971.

18. Kernighan, B. W., "Programming in C - A Tutorial", Bell Laboratories, 1974.

19. McEwen, H. E. (ed.), Management of Data Elements in Information Processing, National Technical Information Service, 1974.

20. David W. Taylor Naval Ship Research and Development Center Report 4751, "User Interface to Database Management Systems", by D. K. Jefferson.

21. Richtie, D. M., On the Security of UNIX, Bell Laboratories, 1975.

22. Haberman, A. N., Introduction to Operating System Design, p. 76-79, Science Research Associates, 1976.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexanderia, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science. Naval Postgraduate School Monterey, California 93940	1
4.	Asst. Professor Gerald L. Barksdale, Jr. Code 52Ba Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
5.	LT Lyle V. Rich, SC, USN, Code 52Rs Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6.	Capt. John Edward Howard, USMC 238 Erskine Place San Antonio, Texas 78201	1